Títol: Control de presentacions mitjançant gestos

Volum: 1/1

Alumne: David Cuenca Aubets

Director/Ponent: Manel Frigola Bourlon

Departament: ESAII

Data: 22/06/10

DADES DEL PROJECTE

Títol del Projecte: Control de presentacions mitjançant gestos

Nom de l'estudiant: David Cuenca Aubets

Titulació: Enginyeria Informàtica (pla 2003)

Crèdits: 37,5

Director/Ponent: Manel Frigola Bourlon

Departament: ESAII

MEMBRES DEL TRIBUNAL *(nom i signatura)*

President: Joan Climent Vilaró

Vocal: Marta Arias Vicente

Secretari: Manel Frigola Bourlon

QUALIFICACIÓ

Qualificació numèrica:

Qualificació descriptiva:

Data:

A Montse, Antonio, Jordi i Sara, per la seva estima
i suport; a elles especialment per donar-me
empenta.

Índex

Índex de figures

Capítol 1

Introducció

1.1 Objectiu

Aquest projecte té com a objectiu desenvolupar un sistema de control remot de presentacions mitjançant gestos com una alternativa natural als comandaments inalàmbrics, per fer-lo servir en qualsevol sala de presentació o classe, on un ponent mostri diapositives per guiar una explicació, fent ús d'un projector connectat a un ordinador. Però a diferència dels mètodes habituals per controlar l'ordinador, amb aquest sistema es podrà comanar l'aplicatiu de presentacions fent signes amb les mans. El principal avantatge d'aquest sistema roman en la naturalitat en realitzar les comandes i en el gran nombre d'ordres diferents que es poden llençar: tantes com gestos es sigui capaç de reconèixer.

1.2 Motivacions

No s'ha pretés resoldre cap problema no solucionat, doncs les presentacions ja es poden controlar remotament mitjançant comandaments, però, de la mateixa manera que al món de les videoconsoles la interacció gestual ha tingut un gran impacte canviant la manera amb que l'usuari controla el joc, es vol fer possible el control de les presentacions d'una altra forma, mitjançant gestos, gràcies a un programa que interpreta les accions que l'usuari vol invocar. Aquest programa s'alimenta de les imatges captades per una càmera connectada al mateix ordinador que executa l'aplicatiu de presentacions i s'ajuda de tècniques de visió per computador per reconèixer els gestos de l'u-

suari, provocant com a resposta a cadascun d'ells diferents events d'entrada al sistema operatiu.

Així doncs es vol fer més natural la interacció amb l'ordinador a l'hora de fer presentacions, però cal remarcar que les mateixes tècniques apreses també es podrien aplicar en qualsevol altre àrea on es requereixi invocar comandes de forma remota, com per exemple en el control d'aparells multimèdia al saló o pel guiatge de robots mòbils. D'aquesta manera, en un futur en comptes de tenir un comandament per a cada artifici, al que l'usuari s'ha d'adaptar, es podrien fer servir els mateixos gestos naturals per a cada comanda (esquerra, dreta, para, etc.) i adaptar-hi els diferents aparells a aquest paradigma, per així controlar-los tots d'una forma natural.

Capítol 2

Conceptes previs

2.1 Antecedents

Han passat 30 anys des de les primeres investigacions en l'àrea de les interfícies d'usuari basades en el control gestual. Una de les primeres investigacions notables en aquesta àrea és la que Richard A. Volt va realitzar al MIT a l'any 1980 [4] en un treball titulat "*Put-That-There: voice and gesture*" en el que es va desenvolupar una interfície on l'usuari podia fer servir en conjunt la veu i el moviment de les mans per moure objectes a una pantalla. Des de llavors la recerca en aquesta àrea ha sigut molt prolífica portant a realitat diverses aplicacions en àmbits com: l'entreteniment, el control d'aplicacions informàtiques, d'electrodomèstics, robots mòbils, la teleassistència, l'ajut a persones grans i discapacitats, la rehabilitació, el reconeixement del llenguatge de signes, etc. Els gestos han sigut capturats fent servir raigs infrarojos, guants electrònics, penjolls, càmeres fixes, visió dual, etc. En l'àmbit comercial els dispositius que han tingut més èxit han estat a la indústria dels videojocs, però el reconeixement d'objectes també ha atret l'atenció de la indústria de l'assistència a malalts, discapacitats i gent gran [3].

En aquest PFC s'ha pretés desenvolupar un aplicatiu d'escriptori que permeti a l'usuari pitjar virtualment tecles (o combinacions d'aquestes) fent gestos amb les mans per controlar remotament presentacions, analitzant amb tècniques de visió per computador les imatges captades per una càmera web. S'han fet molts projectes semblants a aquest i s'ha tingut a mà totes les tecnologies i tècniques necessàries en un grau prou madur per fer aquest PFC viable. Podem dir, doncs, que a Internet es poden trobar diferents projectes

en els que: o bé s'han aplicat les mateixes tècniques que s'han emprat en aquest PFC per altres aplicacions, o bé s'ha perseguit el mateix objectiu (el control de presentacions) mitjançant altres aproximacions al problema. Així per exemple Chen et al.[6] i Barczak et al.[1] han publicat articles en els que s'apliquen els classificadors de característiques de tipus Haar en el reconeixement de gestos, la tècnica principal en la que es basa aquest PFC. Baudel, T. i Beaudouin-Lafon, M. van presentar al 1993 [2] un sistema de control de presentacions que feia ús d'un guant VPL DataGlove^{TM} connectat a un ordinador, amb el que, de la mateixa manera que en aquest PFC, els gestos es reconeixen mitjançant un algorisme que funciona a temps real i que deixa prou potència de CPU per fer funcionar el programari de presentacions a la mateixa màquina. Al grup de recerca *Mobile Machines and Vision Laboratory* (MMVL) de la Sheffield Hallam University, UK, han implementat en Ruby amb una mica menys de 1000 línies de codi un sistema interactiu de presentacions anomenat RMando[1], que amb l'ajut d'una càmera *firewire* detecta la invasió de la mà de l'usuari a l'àrea de la pantalla i permet controlar la presentació movent la mà amunt i avall pels marges de les diapositives.

[1] RMando - Interactive Presentation Software - MMVL
`http://vision.eng.shu.ac.uk/mmvlwiki/index.php/Interactive_Presentation_Software`
(darrera visita el 09/06/10)

2.2 Conceptes base

2.2.1 El color

El color és una rica i complexa experiència causada per la resposta del nostre sistema de visió a diferents longituds d'ona de la llum. Només una estreta secció de l'espectre electromagnètic és visible pels humans, amb una longitud d'ona λ compresa entre els 380 nm i els 740 nm. La intensitat de la irradiació per diferents longituds d'ona λ canvia, i aquesta variació és expressada en un espectre de potència *S(λ)*.

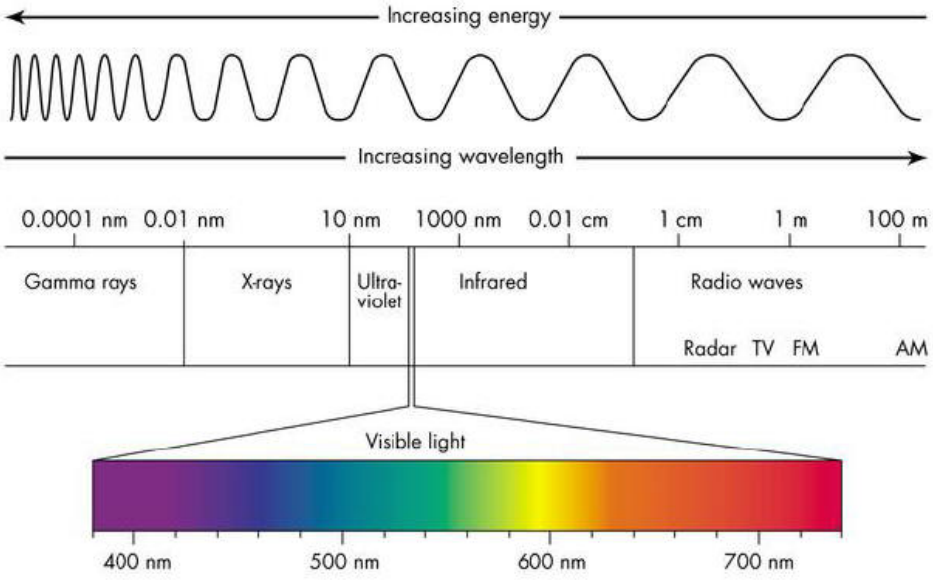


Figura 2.1: Espectre visible

Per una comunicació clara i precisa quan parlem de colors cal fer servir una terminologia ben definida que eviti ambigüitats. L'estàndard *de facto* per a la caracterització dels colors prové de l'*International Lighting Vocabulary*, publicat pel CIE el 1921 (*Commission Internationale de l´Eclairage*, encara en actiu a Laussane, Suïssa). En aquest document s'explica que per especificar amb precisió un color cal tenir en compte aquestes definicions [12]:

- Brillantor: sensació que indica si un àrea sembla emetre més o menys llum.

- Lluminositat: brillantor d'una àrea jutjada en relació a una àrea il·luminada similarment que es percebi com a blanca.
- To: sensació que indica si un àrea és similar al vermell, groc, verd o blau o a una proporció d'aquestos.
- Coloritat: sensació per la que un àrea té un major o menor to.
- Croma: la coloritat d'un àrea jutjada com la proporció de la brillantor respecte d'una àrea il·luminada similarment que es percebi com a blanca.
- Saturació: coloritat d'una àrea jutjada en proporció a la seva pròpia brillantor.

A vegades es confonen els termes de brillantor i lluminositat, però la lluminositat és un concepte que es defineix en relació a un altre color. Per posar un exemple: un paper de periòdic sembla gris al costat d'un full d'oficina. Si els compares dins d'una habitació o fora, a la llum del sol, la lluminositat (relativa) no canvia, però les seves brillantors sí. La mateixa distinció la trobem també entre la coloritat i el croma; quan les condicions de llum canvien el croma tendeix a mantenir-se constant. No és així amb la coloritat: en un dia assolellat veiem les coses amb més coloritat, mentre en un dia ennuvolat tot sembla tenir-ne menys. El cas de la saturació és diferent, doncs no tipifica un fenomen relatiu entre colors, sinó relatiu a la brillantor de si mateix. Així, els cossos presenten una saturació constant per tots els nivells de luminància, excepte quan aquesta és molt alta.

2.2.1.1 El color dels cossos

Hi ha dos fenòmens físics predominants que descriuen què passa quan un cos és irradiat. El primer, la reflexió de la superfície, que rellança l'energia entrant d'una manera similar a un mirall. L'espectre de la llum reflexada es manté igual que la il·luminant i és independent de la superfície –així, els metalls brillants 'no tenen color', només reflexen el color d'altres objectes. El segon, la reflexió del material, en la que la energia és difosa dins el material i és reflexada aleatòriament des del pigment intern de la matèria. Aquest mecanisme és predominant als materials dielèctrics com el plàstic o les pintures. La figura 2.2 il·lustra la reflexió de la superfície i la del propi cos. Els colors

de cert objecte estan causats doncs per les propietats de les partícules del pigment que absorbeixen certes longituds d'ona dins l'espectre de longituds d'ona entrants.

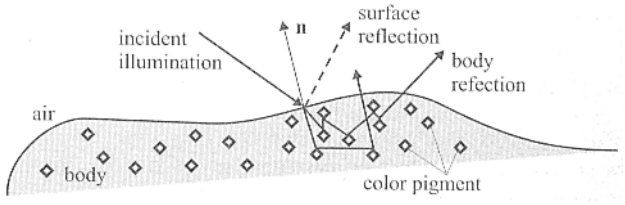


Figura 2.2: Reflexió del cos i la superfície

2.2.1.2 Com percebem els colors

L'evolució ha desenvolupat un mecanisme indirecte de captació del color pels humans i altres animals. Tenim a l'ull tres tipus de sensors receptius a la longitud d'ona de la irradiació entrant, d'aquí el terme tricromacia. Els receptors sensibles a la retina humana són els cons. Els altres receptors sensibles a la retina són els bastons que es dediquen a la percepció monocromàtica en condicions de poca llum ambiental. Els cons estan categoritzats en tres tipus, basats en la longitud d'ona a la que són sensibles: S (*short* (curts)) amb una sensibilitat màxima als $\approx$ 430 nm, M (*medium* (migs)) als $\approx$ 560 nm i L (*long* (llargs)) als $\approx$ 610 nm.

Però no tots els tipus de cons tenen la mateixa sensibilitat. La figura 2.3 il·lustra qualitativament les sensibilitats relatives dels cons S, M, L. Per poder tenir en compte l'absorció de certes longituds d'ona de la còrnia, lents i pigments interns de l'ull es van prendre les mesures amb una font de llum blanca a la còrnia [17].

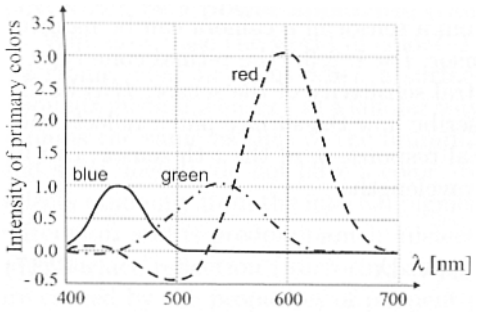


Figura 2.3: Sensibilitat relativa dels cons S, M, L

El fenomen anomenat metàmer és rellevant per entendre com percebem el color. En general en diem metàmer al fet de que dos coses que són físicament diferents es perceben com la mateixa. El vermell i el verd sumats per produir el groc és un metàmer, perquè el groc també pot ser produït per un color espectral, és a dir, amb només una longitud d'ona. El sistema humà de visió és enganyat en la percepció de que el vermell i verd sumats és el mateix que el groc. Així doncs, la visió humana és vulnerable a diferents il·lusions. La percepció del color és influenciada per, apart de l'espectre de l'il·luminant, la interpretació dels colors i l'escena que envolta al color observat (veieu la Figura 2.4). A més, l'adaptació de l'ull a les condicions de llum canviants no és molt ràpida i la percepció dels colors se n'hi veu afectada.

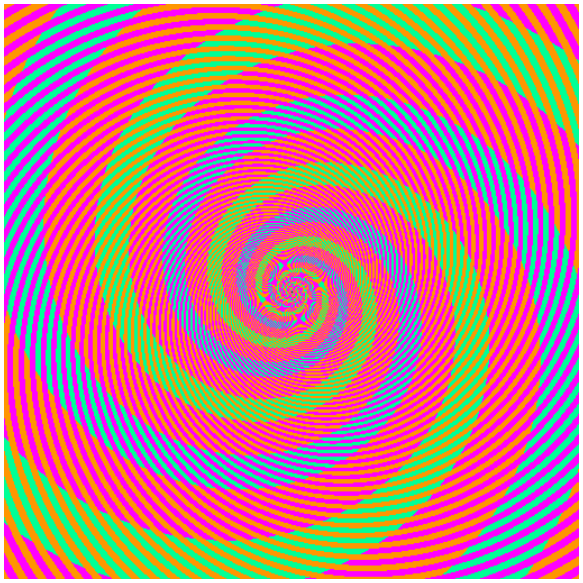


Figura 2.4: Espirals verds i blaves que en realitat són del mateix color.

2.2.1.3 Els models de color

Els models de color van ser introduïts com una abstracció matemàtica que permet expressar els colors com a tuples de números, generalment tres o quatre valors de components cromàtics. Un sol model de color pot tenir una o més instàncies d'espai de color corresponents, depenent de si es tracta de models absoluts o relatius. Per exemple, l'RGB és un model de color que no fixa els valors dels seus colors primaris i existeixen diferents instàncies d'espai de color d'aquest model, com ara l'sRGB o l'Adobe RGB.

2.2.1.3.1 El model CIE XYZ 1931

Es tracta d'un model de color absolut, on no hi ha ambigüitat en la definició dels colors, i amb el que, per tant, només tenim associat un espai de color. Va néixer al 1931, quan, motivat per la premsa i el desenvolupament de les cintes a color, el CIE va publicar aquest estàndard tècnic.

L'estàndard *XYZ* està donat per tres llums imaginàries X=700.0 nm, Y=546.1 nm, Z=435.8 nm i per les funcions d'igualació $X(\lambda)$, $Y(\lambda)$ i $Z(\lambda)$ i compleix aquests tres requisits:

- Les funcions d'igualació XYZ han de ser no negatives;
- El valor d'$Y(\lambda)$ ha de coincidir amb la brillantor (luminància);
- Es realitza una normalització per assegurar que la potència corresponent a les tres funcions d'igualació és la mateixa (és a dir, l'àrea sota les tres corbes és igual)

Les funcions d'igualació són la descripció numèrica de la resposta cromàtica de l'observador. Els valors resultants es mostren a la figura 2.5. El subespai o gamma dels colors perceptibles pels humans es mostra a la figura 2.6

Els valors triestímul del CIE XYZ es poden fer servir per descriure qualsevol color, però és molt poc uniforme perceptualment, de manera que no és apropiat per manipulacions quantitatives que tractin la percepció de color i és rarament usat en aplicacions de processat d'imatges. Encara i això té un paper significant al processament d'imatges ja que altres espais de color es poden derivar d'ell mitjançant transformacions matemàtiques. L'ús

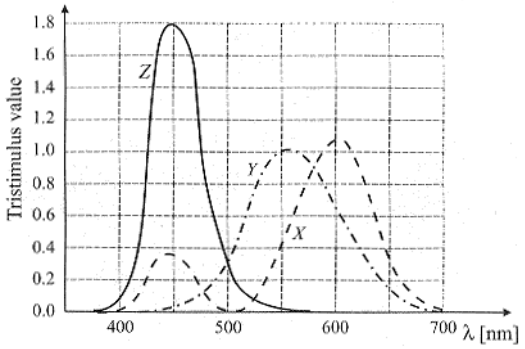


Figura 2.5: Funcions d'igualació de colors per a l'estàndard CIE de 1931

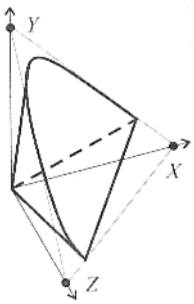


Figura 2.6: Subespai d'XYZ corresponent als colors percebibles pels humans

principal que es dóna a aquest espai de color és pel mesurat de colors (colorimetria). El motiu és perquè es tracta d'un espai de color absolut, en el que els colors són inambigus, de manera que les interpretacions dels colors a l'espai són colorimètricament definits sense fer referència a factors externs, en contraposició als espais de color relatius, que es poden entendre més com una recepta on s'especifica la quantitat de cada ingredient i on la variació d'aquests pot canviar el resultat.

2.2.1.3.2 El model RGB

El model de color RGB és el més conegut de tots. Te el seu origen en la tecnologia de tubs de raigs catòdics on era convenient per descriure el color com una addició de tres raigs, de colors vermell, blau i verd. Encara que és el sistema més intuïtiu de tots, representa un seriós inconvenient: als seus tres valors es mescla el to, la saturació i la coloritat, i això el fa un model poc uniforme. És un exemple d'estàndard relatiu, en el que els colors primaris no estan fixats, així que la percepció del mateix triplet RGB pot canviar entre dispositius. Per això hi ha diferents instàncies de l'espai RGB que poden variar en les matrius de transformació cap a XYZ. Una de les transformacions d'RGB a XYZ és:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 3.24 & -1.54 & -0.50 \\ -0.98 & 1.88 & 0.04 \\ 0.06 & 0.20 & 1.06 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix},$$
$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.41 & 0.36 & 0.18 \\ 0.21 & 0.72 & 0.07 \\ 0.02 & 0.12 & 0.95 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}. \tag{2.1}$$

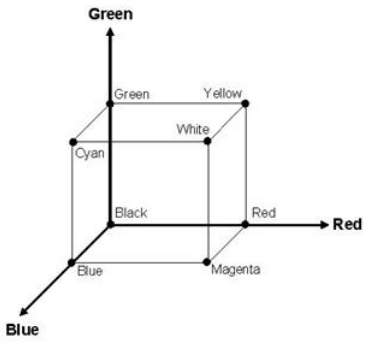


Figura 2.7: Espai de color RGB

2.2.1.3.3 El model HSV

Aquest nom respon a les sigles dels vocables to, saturació i valor en anglès –la V també es sol canviar per la B de brillantor. Els artistes el fan servir sovint perquè és més proper a la seva manera de pensar i treballar, ja que fan servir de tres a quatre dotzenes de colors caracteritzats pel to. Els pintors també volen colors de diferents saturacions, pel que mesclen els colors de la seva paleta amb blanc o negre per aconseguir-ho.

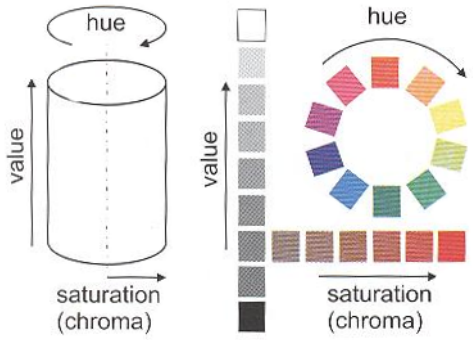


Figura 2.8: Espai de color HSV

L'HSV desacobla la informació de la intensitat del color, mentre el to i la saturació es corresponen a la percepció humana, fent així aquesta representació molt útil per desenvolupar algoritmes de processament d'imatges.

2.2.1.3.4 El model CIE Lab

Consta d'una dimensió L de *lightness* (brillantor) i dos altres a i b on els a positius són vermells, els negatius verds, els b positius són grocs i els negatius blaus (veure Figura 2.9). És un model raonablement uniforme perceptualment. Això vol dir que la distància euclídea entre dos colors a l'espai CIE Lab te una correlació molt forta amb la percepció humana del color. La percepció que tenim del color dels objectes és més un concepte subjectiu que un fenomen físic, així que, una representació del color similar a la sensitivitat humana hauria d'ajudar a la tasca de segmentació per color.

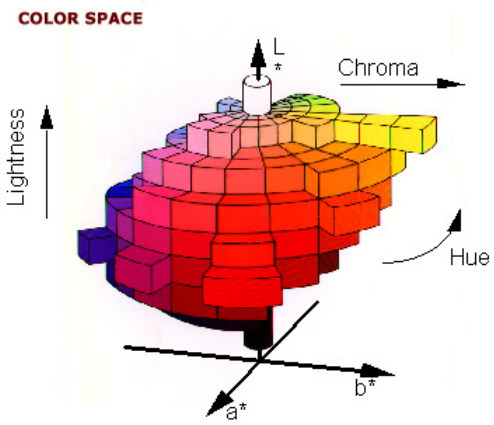


Figura 2.9: Espai de color CIE Lab

2.2.2 La visió per computador

Fer que els ordinadors entenguin el que hi ha a una imatge no és una tasca fàcil. Quan una persona interpreta una imatge el seu coneixement previ i la seva experiència és primordial per poder entendre el que veu. La intel·ligència artificial ha treballat durant dècades per donar als ordinadors la capacitat de entendre observacions, però encara que el progrés ha sigut importantíssim l'habilitat pràctica de les màquines per entendre una escena es manté molt limitada. És per això que les aplicacions de visió per computador treballen sempre en contexts determinats i controlats, sempre coneguts per qui les ha programat.

Les tècniques de visió per computador es valen d'una sèrie d'operacions per aconseguir interpretar imatges en contexts acotats; la captura de la imatge, la seva millora, la detecció de contorns, la identificació de regions, l'anàlisi de formes, de textures, la predicció i el modelat del moviment, el reconeixement d'objectes... són tasques típiques de tot sistema de visió per computador. Aquestes tasques no tenen perquè aplicar-se totes ni tampoc tenen perquè fer-se seqüencialment, però la seva varietat posa de manifest que és necessari un coneixement ampli de la disciplina per dissenyar amb èxit un sistema que interpreti imatges. Un sistema de visió per computador és doncs un sistema en el que partint d'una escena a capturar es van obtenint diferents representacions d'aquesta reduint-ne la complexitat, extraient-l'hi característiques i comprenent-hi el contingut mitjançant algorismes molt diversos. A la Figura 2.10 es mostren quatre nivells de representació apropiats

per resoldre un problema d'anàlisi d'imatges en el que certs objectes han de ser detectats i classificats. Les diferents representacions que s'hi van obtenint estan pintades de gris.

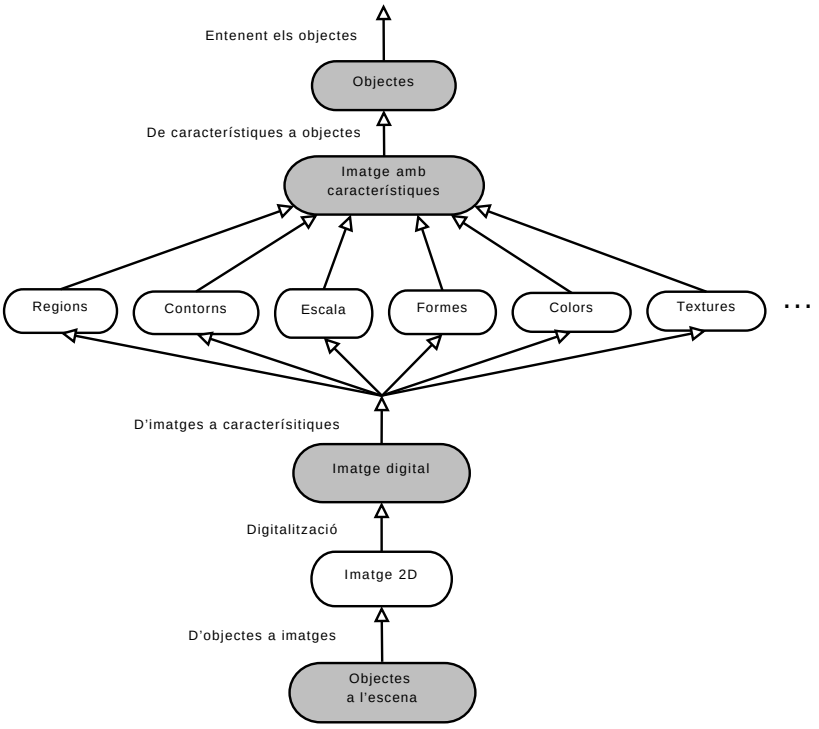


Figura 2.10: Sistema de visió per computador

Capítol 3

Requisits

El sistema a desenvolupar ha d'aprofitar el mateix ordinador que fa servir el ponent per fer córrer l'aplicatiu de presentacions i no ha de requerir cap aparell especial connectat a l'ordinador, tret d'una *webcam,* que en el cas dels ordinadors portàtils pot ser la que normalment porten integrada damunt la pantalla. S'ha d'implementar un programa que mitjançant tècniques de visió per computador permeti al ponent enviar comandes al sistema operatiu de forma remota. Per això el programa haurà de comportar-se com un dispositiu d'entrada virtual que funcioni mitjançant gestos que haurà de poder-se executar simultàniament a l'aplicatiu de transparències, analitzant en segon pla les imatges captades per la *webcam* per reconèixer-hi una sèrie de gestos prefixats que traduirà en events de teclat que es llençaran al sistema operatiu. No es necessita que els gestos a reconèixer siguin dinàmics, dit d'una altra manera, s'hauran de detectar poses concretes de la mà mantingudes durant un curt però significatiu espai de temps, per distingir-les fàcilment de posicions instantànies que la mà del ponent pugui fer en el moviment habitual de l'acte de comunicació.

El programa haurà de ser capaç de reconèixer el gestos de qualsevol usuari que es posi davant la càmera, sigui qual sigui la seva raça. No es demana, però, que l'aplicatiu reconegui els gestos quan aquests apareguin amb un fons amb colors semblants al de la pell, ja que s'és conscient de la dificultat que comportaria aquest fet per qualsevol sistema de visió per computador. De tota manera sí que es vol que el sistema sigui capaç de funcionar en diferents entorns de llum, natural, incandescent o fluorescent i es pressuposa que el ponent no fa la presentació des de la foscor. L'usuari podrà portar tant

màniga llarga com curta.

És necessari que els usuaris del programa tinguin una eina per configurar el programa de manera que aquest pugui reconèixer altres gestos, i també ha de ser possible canviar els events de teclat que hi ha associats a cadascun d'aquests. Cal remarcar que es vol tenir la possibilitat d'associar events de teclat diferents per quan un mateix gest es realitza amb la mà esquerra o amb la dreta.

Capítol 4

Disseny

Per evitar que el programa confongui la comunicació corporal normal del ponent amb ordres s'ha pensat en un mètode per fer prou explícita l'ordre: fer que l'usuari miri cap a càmera en el moment que faci el signe amb la mà per davant del seu tronc. S'ha triat aquest *modus operandi* perquè així s'evitaran fàcilment les equivocacions, ja que només s'hauran d'interpretar gestos estàtics que es trobin exclusivament per sota de la cara de l'usuari. D'aquesta manera s'aconsegueix crear un entorn controlat on trobar l'objecte en qüestió (en aquest cas una mà fent una posa concreta), requisit indispensable en qualsevol aplicació de visió per computador.

Per arribar a detectar amb èxit un gest es realitzaran primer una sèrie de passos que asseguraran aquest entorn controlat: primer es filtraran els colors de la pell, esborrant de la imatge tot allò que no sigui necessari per reconèixer una mà, cosa que facilitarà la tasca de detecció de gestos; en segon terme es localitzarà la cara del ponent i es retallarà per sota d'aquesta l'àrea corresponent al seu tronc i finalment es cercaran els gestos dins d'aquesta àrea.

4.1 Reconeixement de gestos

S'ha dissenyat un sistema en el que es fan servir classificadors d'objectes basats en característiques de tipus Haar per reconèixer els gestos que fa el ponent. Aquesta tècnica, que s'explica en detall al punt 6.3, funciona com un mecanisme que detecta objectes en quant aquests són focalitzats. Els classificadors de gestos s'obtenen mitjançant un entrenament en el que gràcies a una àmplia base d'imatges on apareix el gest en qüestió s'obté una mena d'arbre decisional que permet discernir quan el gest hi és o no dins d'un requadre.

L'algorisme dissenyat és el següent: primer es filtren els colors de la pell, esborrant tot allò que no s'hi correspongui. Després es localitza la cara del ponent a la imatge amb l'ajuda d'un classificador basat en característiques de tipus Haar entrenat per detectar cares de forma frontal. Un cop localitzada la cara es retalla l'àrea per sota d'aquesta corresponent al tronc de l'usuari. Aquesta imatge i la seva imatge mirall es passen al mòdul de reconeixement de gestos, que funciona també a base de classificadors de característiques de tipus Haar. El motiu pel qual es fa servir una imatge girada és per estalviar-se el haver de crear un primer classificador per quan el gest es fa amb la mà dreta i un segon classificador per quan aquest mateix gest es fa amb l'esquerra. Com que les persones tenen simetria vertical la imatge girada d'un gest fet amb la mà esquerra es podrà reconèixer amb un classificador entrenat amb la mà dreta. Per evitar falsos positius els gestos no provoquen un event de teclat en cada instant en que es detecten, sinó que es van comptant i temporitzant les deteccions de manera que només es disparen events de teclat quan els gestos es sostenen durant un espai de temps significatiu.

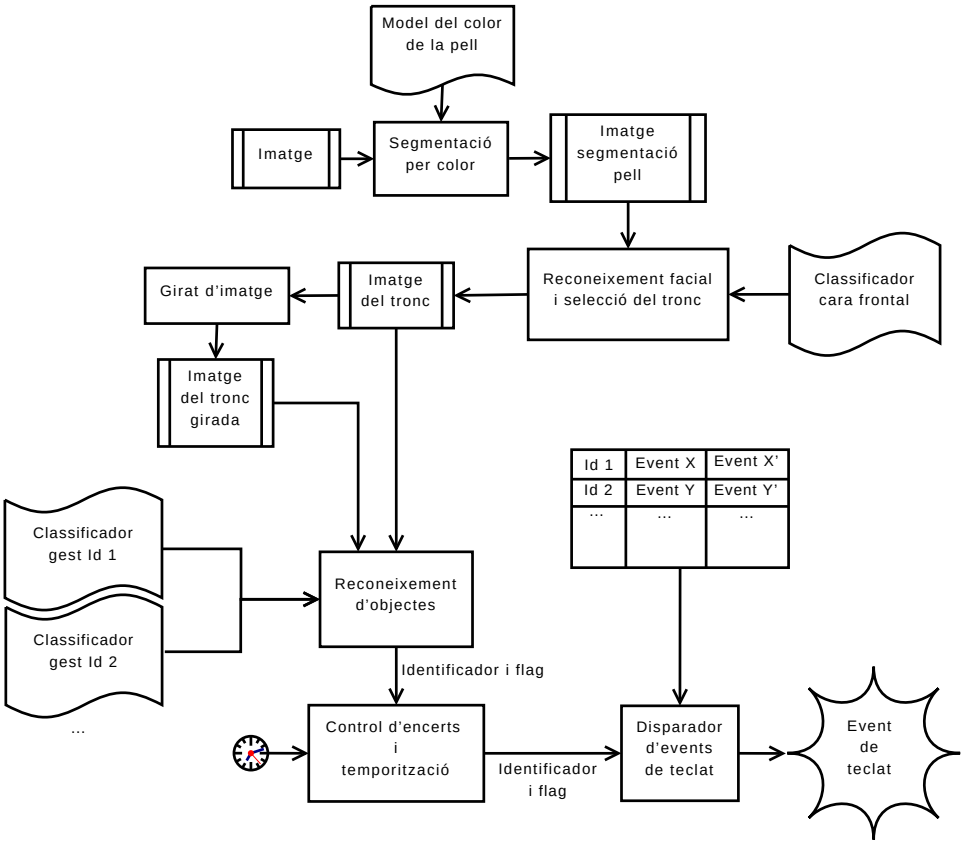


Figura 4.1: Reconeixement de gestos

Mòduls

- **Segmentació per color:** haurà filtrar aquells punts de la imatge que no tinguin un color semblant al de la pell. Rep com entrada les imatges de càmera i el model del color de la pell i dóna com a sortida una imatge segmentada que només conté colors semblants al de la pell.

- **Reconeixement facial i selecció del tronc:** aquest mòdul localitzarà la cara del ponent i retallarà el tronc, que hi està per sota. Té com a entrada la imatge segmentada amb només colors semblants al de la pell, un classificador en cascada basat en característiques de tipus *Haar* que identifica cares frontals i retorna una imatge del tronc del ponent.

- **Girat d'imatge:** aquest mòdul es dedica a capgirar la imatge del tronc del ponent en el seu eix vertical. D'aquesta manera un gest realitzat amb la mà esquerra es veurà com s'hi s'hagués fet amb la dreta i viceversa. Té com a entrada la imatge del tronc original i dóna com a resposta la mateixa imatge girada pel seu eix vertical.

- **Reconeixement d'objectes:** haurà de trobar un gest a les imatges del tronc tant les que estan girades com las que estan del dret. Rep com entrada la imatge del tronc (girada o no) i una llista de classificadors en cascada basat en característiques de tipus *Haar* que identifiquin diferents signes, retorna l'identificador del gest i un marcador (o *flag)* que indica si s'ha trobat el gest a la imatge girada.

- **Control d'encerts i temporització:** haurà de controlar la quantitat de deteccions de cada gest en el temps. Quan s'hagi assolit una xifra significativa de deteccions voldrà dir que l'usuari manté la posició i els encerts no es corresponen a falses deteccions circumstancials. Rep com a entrada els identificadors dels gestos amb el seu *flag* corresponent així es van detectant, i mitjançant un comptatge i temporització d'aquestos decideix quan s'ha detectat un gest, moment en que dóna com a resposta l'identificador del gest que està fent l'usuari.

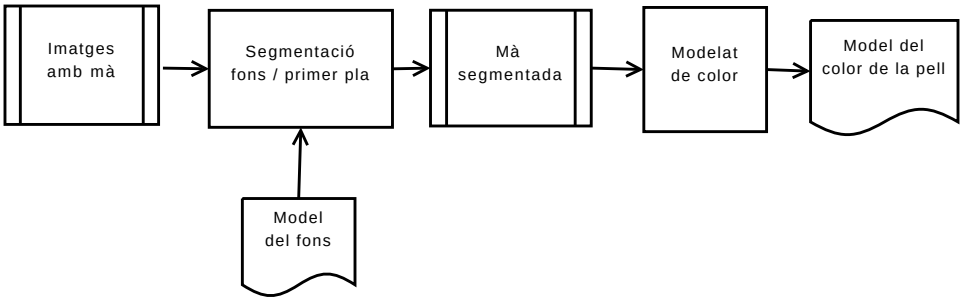
- **Disparador d'events de teclat:** haurà de enviar events de teclat al sistema operatiu quan s'hagi detectat un gest. Quan rep un identificador d'un gest i el *flag* que correspon a si aquest s'ha detectat del dret o girat i cerca a una taula de correspondència gest-event quin event de teclat ha de provocar com a resposta. A l'esquema es representa com a Event X l'event a disparar quan el gest es detecta del dret i Event X' quan aquest es detecta girat (volent dir que s'ha fet amb la mà contraria a la que es va fer servir per l'entrenament del classificador).

4.2 Obtenció del model del color de la pell

Ja s'ha dit que per aportar robustesa al sistema en una primera fase, abans de buscar els gestos, es fa un filtrat dels colors de la pell. Per aconseguir-ho serà necessari disposar d'un model del color de la pell, que es podrà construir i guardar en un fitxer per el seu ús posterior en altres execucions, ara bé, el programa es distribuirà amb un model de color que servirà pel to de pell caucàsic que serà el que es carregarà per defecte. Per adaptar el programa a altres ètnies s'ha dissenyat el sistema que es representa a la Figura 4.2, amb el que l'usuari podrà construir un model del color de la pell i guardar-ho a un fitxer per a usos posteriors.

El sistema funciona de la següent manera: si es sol·licita crear un nou model de color el programa engegarà un assistent amb el que en primer terme es modelarà el fons rere un àrea marcada a la pantalla (4.2a); en segon terme es demanarà que l'usuari mogui el braç per aquesta àrea per fer-n'hi un model del seu color de pell (4.2b).

(a) Modelat del fons



(b) Segmentació de la mà i modelat dels colors de la pell

Figura 4.2: Obtenció del model de color de la pell

Mòduls

- **Modelat del fons:** haurà de crear un model del fons que serveixi per fer després una segmentació de fons / primer pla. S'alimenta d'imatges captades per la càmera i retorna un model del fons.
- **Segmentació fons / primer pla:** haurà d'esborrar el fons de la

imatge mostrant només allò que hi hagi en primer pla, en aquest cas, la mà de l'usuari. S'alimenta d'imatges captades per la càmera, el model del fons i retorna imatges segmentades on només es mostra la mà.

- **Modelat de color:** haurà de modelar la gama de colors de la pell. S'alimenta d'imatges on només apareix la mà de l'usuari i retorna un model de color de la pell.

4.3 Disseny de la interfície gràfica

Finestra principal

En aquesta finestra l'usuari té l'opció de triar entre:

- Engegar el programa amb la configuració per defecte (botó "Start with default settings"):
  - Es farà servir el model de color per defecte (to de pell caucàsic).
  - Es reconeixeran els gestos:
    - esquerra: mà dreta en forma de pistola apuntant a l'esquerra amb l'event de teclat corresponent al pitjat de la tecla Esquerra associat.
    - dreta: el mateix però amb la dreta i amb la tecla Dreta associada.
    - engegar la presentació: mà dreta oberta mostrant el palmell a càmera amb la tecla F5 associada.
- Modificar la configuració (botó "Modify settings"): es mostrarà la finestra "Modificar configuració".

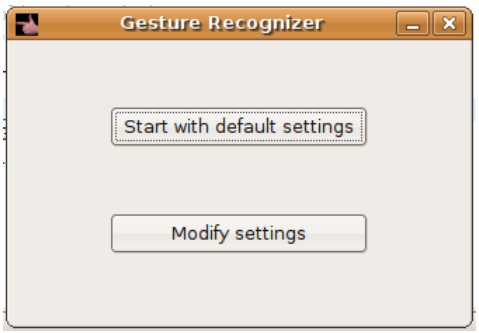


Figura 4.3: Finestra principal

Finestra Modificar configuració

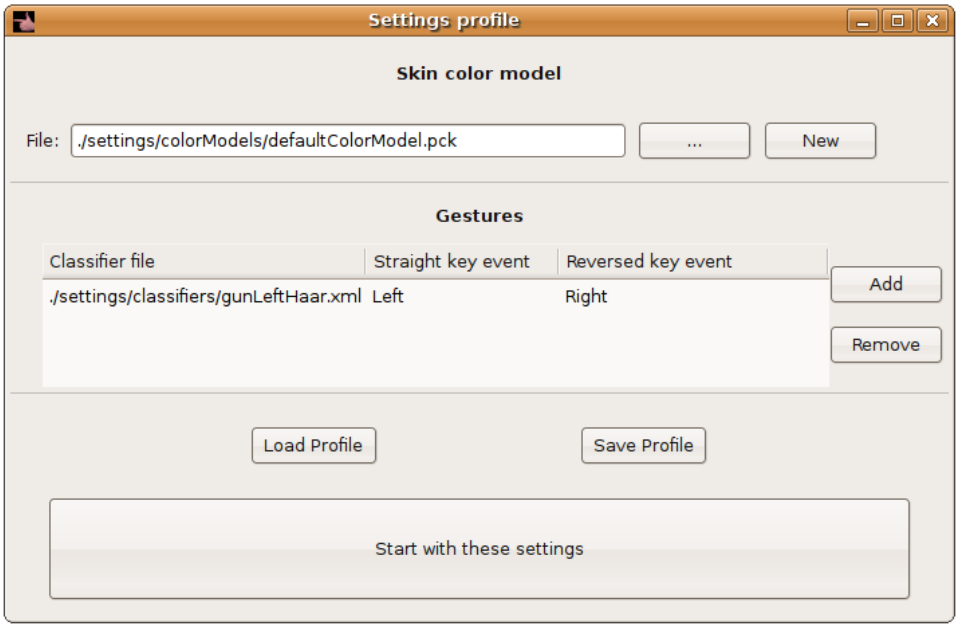


Figura 4.4: Finestra modificar configuració

En aquesta finestra l'usuari té l'opció de:

- Triar un model de color:
  - Seleccionant un model de color existent (botó "..."): cas en el que s'engega un navegador de fitxers en el que es poden triar fitxers del tipus "Model de color (*.pck)"
  - Creant un de nou (botó "New"): cas en el que primer es demana triar un nom de fitxer pel model que es crearà i després s'engegarà l'assistent al que fa referència el punt 4.2.
- Modificar la llista de gestos a reconèixer:
  - Afegint nous gestos a la llista (botó "Add): cas en el que es mostrarà la finestra "Nou gest".
  - Seleccionant un gest de la llista i eliminant-lo (amb el botó "Remove").
- Carregar un perfil (botó "Load Profile"):

  - Es mostra un navegador de fitxers per triar-ne un que contingui un perfil de configuració existent (fitxers de tipus "Perfil de configuració (*.yaml)"

- Desar la el perfil de configuració actual (botó "Save Profile"):

  - Es mostra un navegador de fitxers per triar el nom i la ubicació en el que es vol guardar la configuració actual (és a dir, amb el model de color i la llista de gestos que es mostren a la finestra). Es desarà el fitxer amb l'extensió "yaml".

- Començar amb aquesta configuració (botó "Start with these settings"):

  - Comença el reconeixement de gestos explicat al punt 4.1 amb la configuració que es mostra a la finestra actualment.

Finestra Nou gest

Figura 4.5: Finestra Nou gest

Aquesta finestra es mostra quan es pitja el boto "Add" de la finestra "Modificar Configuració". Aquí l'usuari pot afegir un nou gest a la llista de gestos reconeixibles i especificar quins events de teclat es dispararan en ser detectat. Cal recordar que els classificadors s'aconsegueixen mitjançant un entrenament amb imatges en les que el gest en qüestió sempre es fa amb la mateixa mà. Un sol classificador, però, és útil per reconèixer el gest fet amb la mà contrària: només cal girar la imatge perquè sembli que es fa la mà original. En aquesta finestra l'usuari disposa de:

- Un camp per especificar el fitxer del classificador que detecta el gest ("Classifier file"). I un botó ("...") per engegar un navegador de fitxers per triar-lo.
- Dos files ("Straight gesture key event" i "Reversed gesture key event") on es pot especificar l'event de teclat que dispararà la detecció del gest, una per quan es detecta el gest en la posició amb la que s'ha entrenat el classificador i l'altre per quan el gest es detecta girat, és a dir, quan l'usuari el realitza amb l'altre mà. A cadascuna d'aquestes files trobem:
  - Tres botons de tipus interruptor que indiquen els modificadors (Ctrl - Alt - Shift) associats a l'event de teclat.
  - Un camp sensible, que un cop seleccionat registra la tecla que l'usuari prem al teclat.

A l'exemple de la Figura 4.5 veiem com s'associen els events de teclat "Shift-Tab" i "Ctrl-Alt-d" al gest que detecta el classificador "gestureXClassifier.xml" per quan aquest es detecta del dret o girat (respectivament). Cal remarcar que cada classificador (o fitxer xml) només reconeix directament

un gest, fet amb una mà en concret, però com ja s'ha explicat es fa servir el mateix classificador per reconèixer el mateix gest fet amb l'altre mà gràcies a que el programa analitza per aquest cas la imatge mirall de la original.

Capítol 5

Planificació i costos

5.1 Planificació inicial

En començar el PFC es van planificar 750 hores de treball corresponents als 37,5 crèdits de pes que té el projecte d'EI. Això suposava, doncs, 18 setmanes aplicant-hi 8 hores i 20 minuts diaris de dilluns a divendres. A la Figura 5.2 es mostra la distribució en el temps de les principals tasques que s'esperava fer en aquest projecte. D'altra banda, la càrrega de treball estimada era la següent:

Estudi estat de l'art	25 h.
Comparativa tecnologies a emprar	20 h.
Estudi tecnologies a emprar	60 h.
Estudi teoria	45 h.
Anàlisi requisits	5 h.
Especificació	5 h.
Disseny	15 h.
Implementació prototip	120 h.
Entrenament 1	25 h.
Canvis especificació, redisseny	12 h.
Implementació versió final	100 h.
Implementació interfície usuari	50 h.
Entrenament 2	30 h.
Proves	58 h.
Redacció memòria	140 h.
Revisió memòria 1	20 h.
Revisió memòria 2	20 h.
	750 h.

Figura 5.1: Càrrega de treball estimada

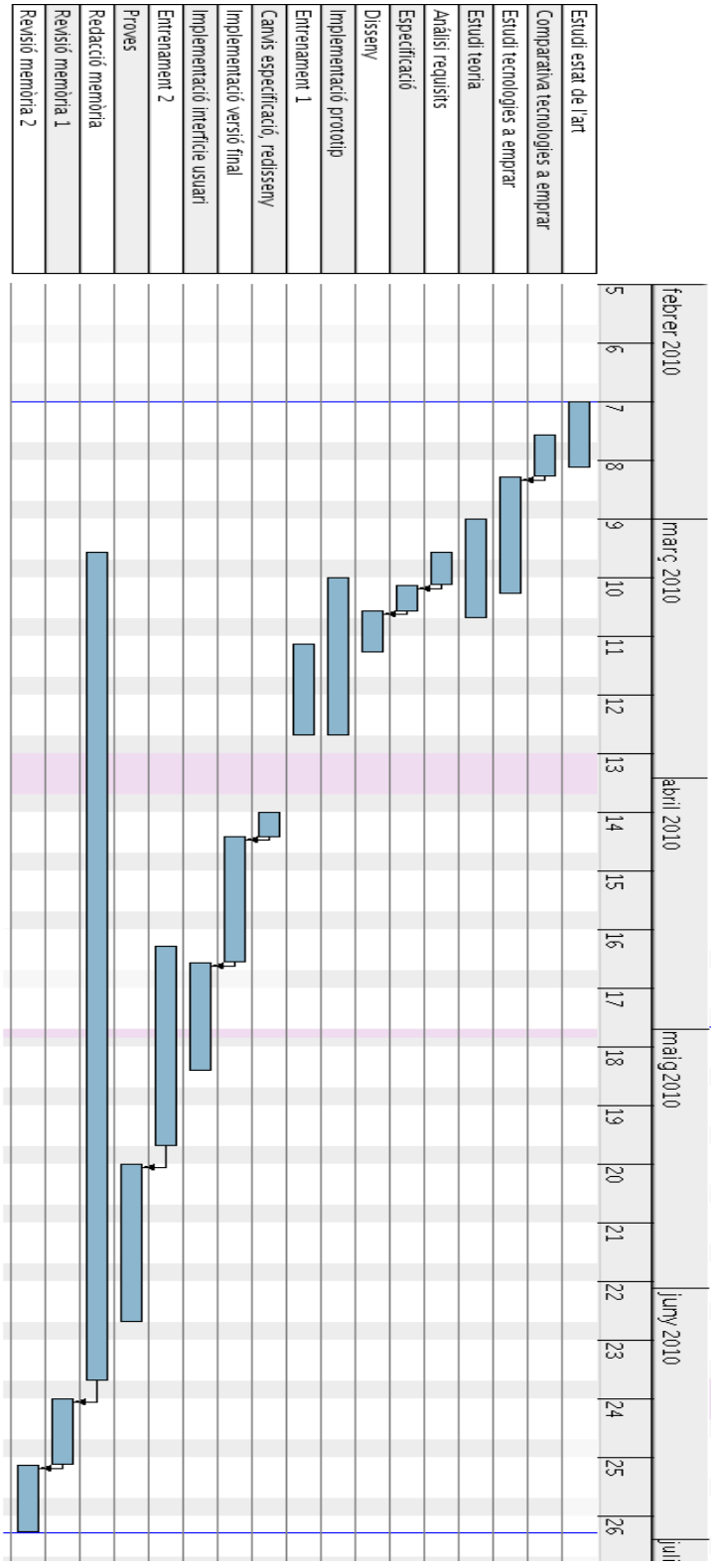


Figura 5.2: Planificació del projecte

5.2 Estudi econòmic previ

L'estudi econòmic previ (veure Figura 5.3) es va basar en aquesta càrrega de treball, contemplant només la mà d'obra d'un enginyer informàtic assumint els rols d'analista i programador. Aquesta estimació contempla també el cost de l'adquisició d'un portàtil pel desenvolupament de l'aplicatiu, l'adquisició d'un ordinador de sobretaula potent per realitzar els entrenaments dels classificadors i la part proporcinal del cost de lloguer d'un despatx tècnic que inclou en el seu preu les despeses en serveis. Finalment s'aplica un marge de benefici, un marge per contemplar els costos de gestió empresarial i l'impost sobre el valor afegit.

Sou analista programador	28.000	€ bruts / any
	36.400	S.S. (+30% aprox.)
	1.700	hores / any
	21	€ / hora
	750	hores projecte
	16.059	**€**
PC per desenvolupament i proves	600	€ portàtil
	3	anys amortització
	83	**€ (5 mesos)**
PC per entrenament classificadors	900	€ sobretaula
	3	anys amortització
	125	**€ (5 mesos)**
Lloguer despatx 4 persones	1.500	€ / mes Tot inclòs (llum, aigua, mobiliari, internet, etc.)
	1.875	**€ / persona (5 mesos)**
	18.142	**TOTAL**
	21.771	+20% marge de benefici
	25.036	+15% costos gestió empresa
	29.042	**Amb IVA**

Figura 5.3: Estudi econòmic previ

Noti's que no es fan càrrecs sobre l'adquisició de llicències de programari, doncs el projecte s'ha desenvolupat íntegrament amb aplicatius i llibreries de codi lliure.

5.3 Desviacions i cost final

Fins ara s'ha presentat el pressupost i la planificació inicial, però, com és habitual en qualsevol projecte, han hagut desviacions i sobrecàrregues que han incidit en el cost real final. Es presenta a continuació una taula on es mostren les càrregues de treball reals en més detall que en la planificació inicial, doncs s'expressen les hores invertides en cadascun dels mòduls explicats al capítol "Disseny" (veure el punt 4) i en altres tasques considerades rellevants que no estaven contemplades a la planificació inicial.

	Planificació inicial	**Dedicació real**
Estudi estat de l'art	25	25
Comparativa tecnologies a emprar	20	30
Estudi tecnologies a emprar	60	100
Instal·lació entorn desenvolupament	0	15
Estudi teoria	45	40
Anàlisi requisits	5	5
Especificació	5	5
Disseny	15	15
Implementació (prototip i versió final)	220	*(veure desglossament)*
Modelat del fons		25
Segmentació fons / primer pla		20
Modelat de color		40
Segmentació per color		20
Reconeixement facial i selecció del tronc		5
Girat d'imatge		1
Reconeixement d'objectes		35
Control d'encerts i temporització		30
Disparador d'events de teclat		35
Altres implementacions fora del disseny original		
Gravació en vídeo		4
Refinat del model de color		15
	Subtotal	**230**
Entrenaments classificadors	55	70
Canvis especificació, redisseny	12	8
Implementació interfície usuari	50	50
Proves	58	10
Redacció memòria	140	170
Revisió memòria 1	20	10
Revisió memòria 2	20	0
Total	**750**	**783**

Figura 5.4: Càrrega de treball real

Com es pot veure la dedicació real al projecte ha sigut d'unes 780 hores i escaig, cosa que escurçaria el marge de benefici en uns 700 €. S'han sacrificat,

però, hores en algunes tasques com la realització de proves i les revisions de la memòria, havent-hi també tasques que han absorbit més o menys temps de l'esperat. A continuació es justifiquen les desviacions més rellevants:

p.i.: planificació inicial
d.r.: dedicació real

- Comparativa tecnologies a emprar [20h. p.i. - 30h. d.r.]: es va valorar l'ús de la llibreria HornetsEye[1] amb la que està implementat RMando (veure 2.1), l'ús de MATLAB i l'ús d'OpenCV entre altres. Es van perdre moltes hores inútilment, ja que HornetsEye va resultar tenir poca documentació i MATLAB no donava suport per la càmera en Linux.

- Estudi tecnologies a emprar [60h. p.i. - 100h. d.r.]: aquesta tasca contempla l'aprenentatge en l'ús de la llibreria OpenCV i del llenguatge de programació Python. També inclou les hores invertides entremig de les tasques d'implementació per aprendre a fer servir el SWIG i altres llibreries que han sigut útils per aquest projecte. De totes aquestes tecnologies no se'n tenia cap coneixement abans de començar el projecte i les hores d'aprenentatge invertides van ultrapassar tota expectativa.

- Instal·lació entorn de desenvolupament [0h. p.i. - 15h. d.r.]: va ser una errada no considerar aquesta tasca en la planificació inicial, doncs la instal·lació i configuració de l'Eclipse, el Python, el compilat de la llibreria OpenCV 2.0 i la configuració del servidor de control de versions, entre altres coses, han absorbit un temps no menyspreable.

- Implementació [220h. p.i. - 230h. d.r.]: la implementació de funcionalitats extra necessàries pel desenvolupament del projecte i la falta d'experiència en projectes d'aquest tipus han penalitzat la inversió d'hores en aquestes tasques.

- Entrenaments classificadors [55h. p.i. - 70h. d.r.]: la tasca d'entrenament contempla també la recopilació d'imatges on els gestos apareixen aïllats i una tediosa feina d'entrenament que la majoria dels cops no era fructífera. S'han tingut molts problemes amb els classificadors en cascada.

- Proves [58h. p.i. - 10h. d.r.]: per una mala planificació o per falta de temps no s'han pogut fer proves quantitatives del funcionament

[1] HornetsEye - Computer Vision for the Robotic Age
`http://hornetseye.sourceforge.net/` (darrera visita el 22/06/10)

del programa, encara que sí una feina contínua de comprovacions així s'anava fent la implementació.

- Redacció memòria [140h. p.i. - 170h. d.r.]: la redacció de la memòria ha portat més temps de l'esperat, en part per haver fet ús de LyX per aconseguir una millor presentació, invertint hores d'aprenentatge en aquest producte que han inflat les hores gastades en aquesta tasca.

Cal remarcar que el cost final que es reflexa en aquest apartat podria experimentar una rebaixa important si la empresa que s'encarregués de desenvolupar aquest projecte considerés les hores emprades en aprenentatge i familiarització amb les tècniques de visió per computador aplicades com una inversió que s'amortitzaria en projectes futurs de la mateixa índole.

Capítol 6

Tècniques de visió per computador emprades

6.1 Sostracció del fons fent servir un *codebook*

Per sostraure el fons d'una imatge primer cal "aprendre" un model d'aquest. Un cop s'ha aprés aquest model es compara amb les imatges subsegüents i llavors el fons conegut s'esborra de la imatge. Els objectes que han quedat a la imatge són presumiblement nous objectes de primer pla. Normalment, en visió per computador, el fons és tot allò que es manté estàtic o fluctuant en el transcurs del període d'interès. Així, les fulles d'un arbre que el vent mou darrere una finestra, les possibles vibracions de la càmera o els canvis de llum que modifiquen una mica la imatge, no han de canviar el que es considera fons.

El mètode de sostracció del fons mitjançant l'ús d'un *codebook* [7] s'inspira en les tècniques de compressió de vídeo creant una estructura de dades que modela les possibles variacions dels valors dels píxels. Un *codebook*, doncs, no és res més que un conjunt de caixes que marquen rangs de valors que ha anat prenent cada píxel. Cada caixa va representant un rang de valors *possibles*, i així que es van trobant petites fluctuacions les caixes creixen per abastar aquests valors propers. Quan es troba un canvi brusc es crea una nova caixa. En el cas de fer servir imatges en color el *codebook* abasta els valors dels píxels en cada canal. La Figura 6.1 il·lustra aquesta explicació.

A la pràctica no és convenient triar l'espai RGB per representar els colors de les imatges a partir de les quals se'n modelarà el fons. És sempre millor

triar un model de color on un eix estigui alineat amb la brillantor. La raó per això és perquè, empíricament, la major part de la variació dels colors de fons tendeix a estar al llarg de l'eix de la luminància i no del color.[5]

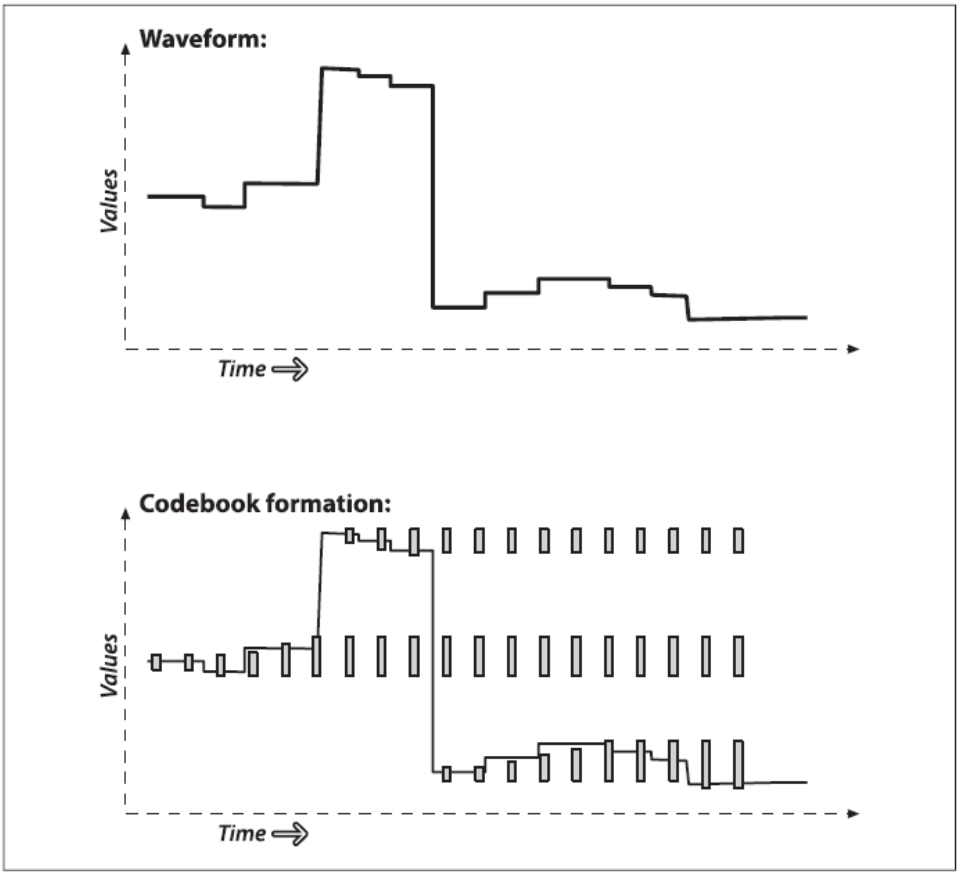


Figura 6.1: Construcció d'un *codebook*

6.2 Modelat del color de la pell

Totes les pells, sense pigments, semblen blanques a la seva base. Els vasos sanguinis propers a la superfície afegeixen un to vermellós. Els carotenoides del greix afegeixen un pigment groc. La melanina dóna una tonalitat sèpia, creada com a resposta als raigs ultravioleta, que en cas de grans concentracions semblen negres. A grans trets el color de la pell és la conseqüència de la reflexió de la llum amb diverses longituds d'ona sobre aquests quatre pigments i també depèn de les reflexions de la seva superfície, que provoca ombres i lluïssors (veure 2.2.1.1). Es tracta doncs d'un fenomen complex

difícil de modelar perquè la gamma de colors resultants no es concentra en cap àrea concreta de cap espai de color (veure 2.2.1.3).

Per modelar el color de la pell, en aquest projecte, s'ha desenvolupat un mètode basat en la idea de taula de cerca normalitzada (*normalized Lookup Table* [15, 18]*)*. Per obtenir un classificador de colors en dos grups, PELL-NO PELL, primer construïm un histograma de la següent manera:

1. Es pren una seqüència d'imatges en les que només es veu un braç o una mà nua sobre un fons negre (obtingudes mitjançant la sostracció de fons explicada en el punt anterior).

2. Es traslladen els colors de cada imatge a l'espai CIE Lab.

3. Per a cada píxel no negre es calcula el triplet $\left(\left\lfloor\frac{a}{n}\right\rfloor, \left\lfloor\frac{b}{n}\right\rfloor, \left\lfloor\frac{l}{n}\right\rfloor\right)$ on (a, b, l) son les coordenades del color del píxel a l'espai CIE Lab i n és un divisor del valor màxim M que podria prendre cadascuna d'aquestes coordenades.

4. Es manté un diccionari de contingut:$\left(\left\lfloor\frac{a}{n}\right\rfloor, \left\lfloor\frac{b}{n}\right\rfloor, \left\lfloor\frac{l}{n}\right\rfloor\right) \rightarrow$ *comptador*

D'aquesta manera aconseguim construir un histograma en forma de diccionari on cada columna representa un subcub de l'espai CIE Lab. Aquestes columnes podran tenir etiquetes en el rang de valors $\left[(0,0,0)\ldots(\frac{M}{n}, \frac{M}{n}, \frac{M}{n})\right]$, representant així des del primer subcub $(0,0,0)$ amb aresta n situat a l'origen de coordenades de l'espai CIE Lab fins al darrer subcub $\left(\frac{M}{n}, \frac{M}{n}, \frac{M}{n}\right)$ d'aresta n situat a l'altre extrem d'aquest espai. Així cada columna de l'histograma representa un comptador corresponent a un conjunt de colors semblants, perquè el CIE Lab és un model uniforme perceptualment (veure 2.2.1.3.4) i per tant dos valors propers en aquest espai seran similars. La figura 6.2 representa esquemàticament com es construeix el model del color de la pell.

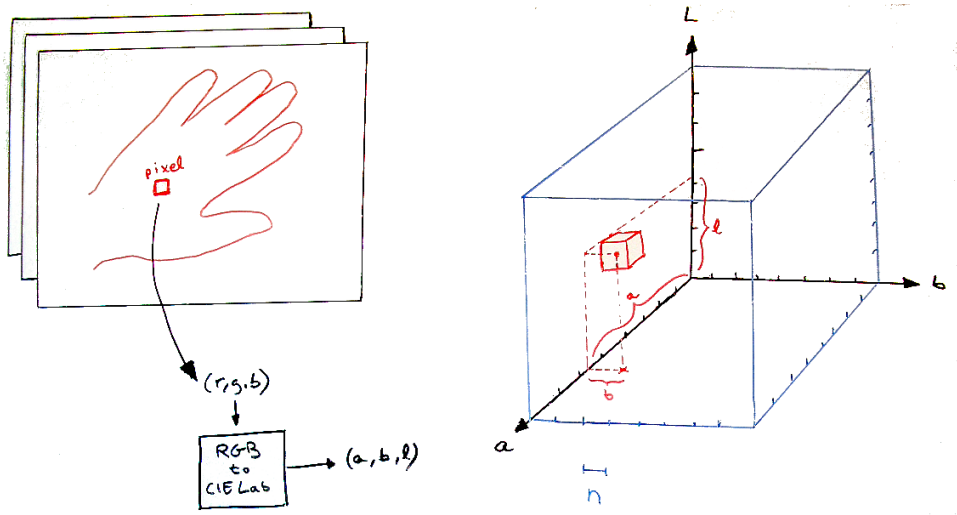
Un cop construït un histograma amb aquest mètode hi haurà columnes amb relativament pocs valors producte d'errors en la segmentació de la mà, i per tant seran colors de fons, que no es consideren pell. Per això l'histograma es normalitza encabint tots els seus comptadors en el rang [0..1] dividint tots els comptadors pel valor màxim, establint després un valor llindar per sota del qual trobarem els comptadors de colors NO PELL.

L'histograma construït servirà per segmentar pel color de la pell, aïllant la mà del ponent i facilitant així la tasca dels classificadors. Per segmentar

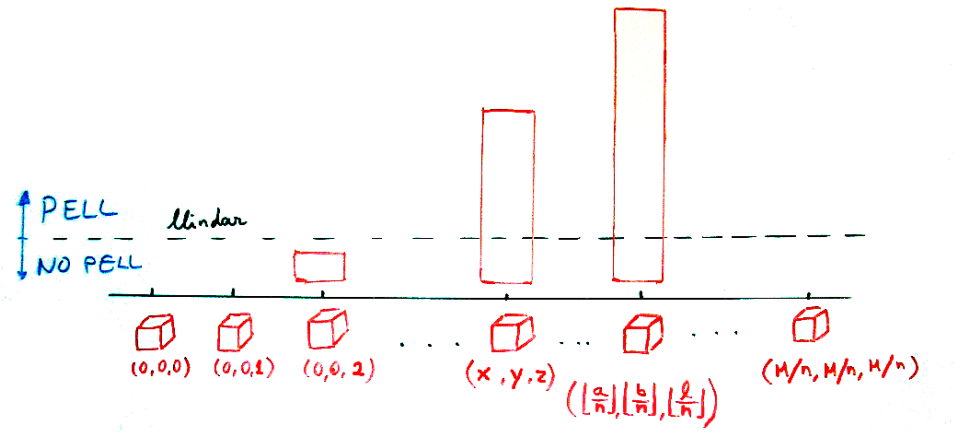
els colors en PELL-NO PELL es procedirà de la següent manera:

1. Es pren la imatge a segmentar i es trasllada a l'espai CIE Lab.
2. Per a cada píxel es calcula el triplet $\left(\left\lfloor \frac{a}{n} \right\rfloor, \left\lfloor \frac{b}{n} \right\rfloor, \left\lfloor \frac{l}{n} \right\rfloor\right)$ on n torna a ser el llarg de l'aresta dels subcubs en que s'ha dividit l'espai CIE Lab.
3. Es mira si el subcub amb etiqueta $\left(\left\lfloor \frac{a}{n} \right\rfloor, \left\lfloor \frac{b}{n} \right\rfloor, \left\lfloor \frac{l}{n} \right\rfloor\right)$ està a l'histograma (diccionari). Si hi és i el seu comptador està per sobre del llindar significarà que es tractava d'un color semblant a d'altres que es corresponien al de la pell (ha caigut dins del mateix subcub) i per tant el considerem *pell* perquè és semblant.

L'augment o disminució del valor que es tria per a n modificarà la granularitat del model. Recordem que n és el llarg de l'aresta de cada subcub amb que subdividim l'espai de color i és, per tant, un divisor del valor màxim M de les coordenades del CIE Lab (en imatges amb 8 bits per color $M = 256$). Quan $n \to 1$ el model té més granularitat, ocupa més espai a memòria (fins al punt de guardar un comptador per color) i dos colors que podríem considerar semblants cauen a subcubs diferents. Quan $n \to M$ el model ocupa menys memòria, però dos colors que podríem considerar molt diferents caurien dins del mateix subcub.



(a) Localització d'un color dins d'un subcub de l'espai CIE Lab



(b) Histograma que modela els colors de la pell

Figura 6.2: Modelat del color de la pell

6.3 Reconeixement d'objectes amb classificadors basats en característiques de tipus Haar

Aquest mètode va marcar un abans i un després a les tècniques de reconeixement d'objectes, sobretot en la detecció de cares. Va ser introduït per C. P. Papageorgiou, M. Oren i T. Poggio el 1998 [10]. Al 2001 P. Viola, M. Jones [16] el van popularitzar millorant la tècnica fent servir imatges integrals i disposant els classificadors en cascada. D'ençà, altres han aportat estudis complementaris [9, 8] i s'ha fet servir amb èxit també en la detecció de gestos [1, 7, 6].

Amb aquest mètode es construeix un classificador en cascada basat en característiques de tipus Haar[1] que es fa servir com a un mecanisme que detecta objectes quan són focalitzats. Per computar ràpidament les característiques de tipus Haar es fa servir el que Viola i Jones van anomenar imatge integral, un càlcul sobre una imatge d'un sol canal amb el que a cada punt (x, y) es guarda la suma dels píxels per sobre i a l'esquerra d'aquest incloent-lo a ell mateix:

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y'),$$

on $ii(x, y)$ és la imatge integral i $i(x', y')$ la imatge original. Fent servir el següent parell de recurrències:

$$s(x, y) = s(x, y - 1) + i(x, y)$$

$$ii(x, y) = ii(x - 1, y) + s(x, y)$$

on $s(x, y)$ és la suma acumulativa de la fila, $s(x, -1) = 0$ i $ii(-1, y) = 0$ la imatge integral pot ser computada d'una sola passada.

Fent servir la imatge integral qualsevol suma rectangular pot ser calculada amb quatre referències. Així, la suma dels píxels dins el rectangle D de la Figura 6.3 és pot calcular com $4 + 1 - (2 + 3)$. On el valor de la imatge integral a la posició 1 és la suma dels píxels al rectangle A, el valor a 2 és $A+B$, el valor a 3 és $A+C$ i el valor a 4 és $A+B+C+D$.

[1] Anomenades així perquè es computen de forma similar als coeficients dels *Haar wavelets transforms*

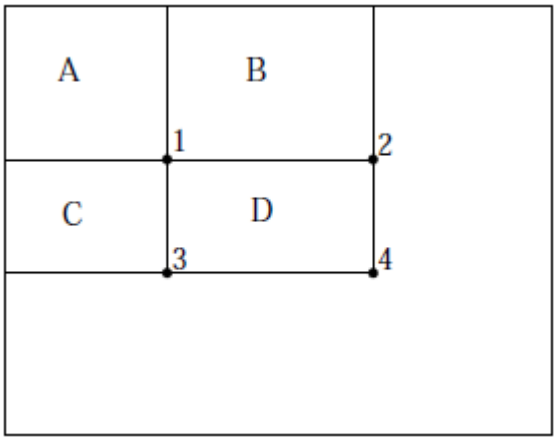


Figura 6.3: Càlcul de la suma dels píxels dins d'un rectangle fent servir la imatge integral

D'aquesta manera, la diferència entre dos sumes rectangulars pot ser calculada, doncs, amb vuit referències. De forma semblant es pot calcular la imatge integral per calcular diferències entre rectangles disposats en un angle de 45°[9]. Aquests càlculs es fan servir per establir els classificadors de característiques de tipus Haar, que descriuen la proporció entre les parts fosques i brillants a dins d'un *kernel*. Cada *kernel* està pensat per detectar o bé marges, punts foscos o línies (veure Figura 6.4).

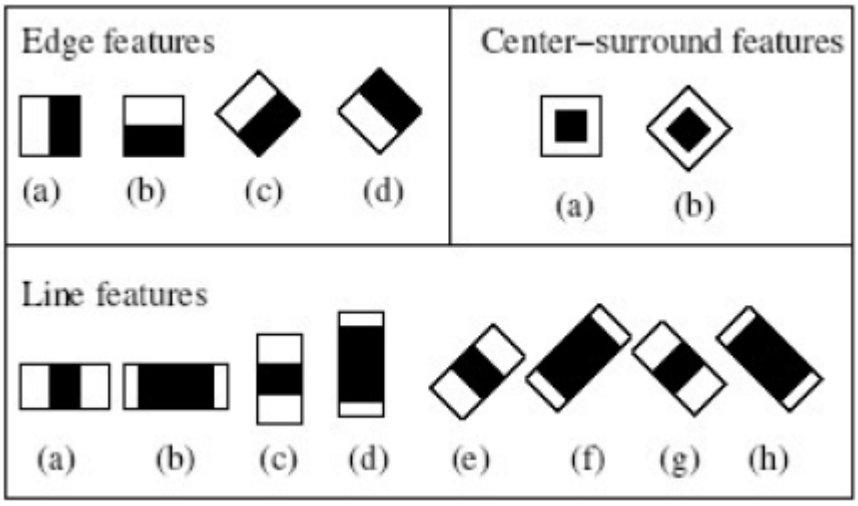


Figura 6.4: Conjunt de *kernels* de característiques Haar

Un conjunt d'aquests *kernels* es pot fer servir per codificar els contrasts

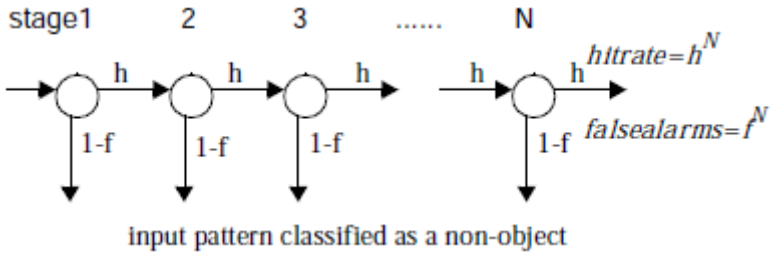


Figura 6.6: Classificador en cascada

amb certa orientació i les seves relacions espacials que exhibeix cert objecte (veure Figura 6.5).

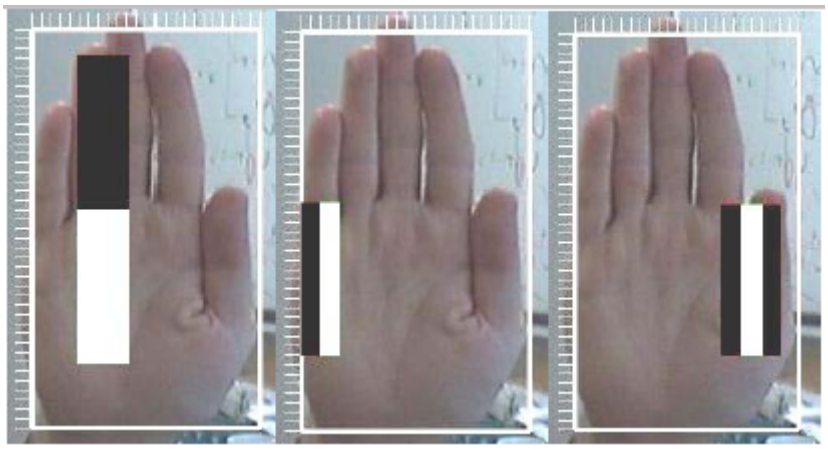


Figura 6.5: Contrastos característics a un gest

Per aconseguir el detector d'un objectes s'entrena un classificador reforçat en cascada amb centenars d'imatges de l'objecte (en aquest projecte una certa postura de la mà) anomenats exemples positius, que s'escalen a la mateixa mida (per exemple 20x20) i exemples negatius d'imatges arbitràries on no hi apareix l'objecte.

Una cascada de classificadors és un arbre decisional degenerat en el que a cada fase un classificador és entrenat perquè detecti casi tots els objectes d'interès rebutjant una fracció dels patrons de tot allò que no ho és (veure Figura 6.6).

El mot reforçat (*boosted*) fa referència a que els classificadors a cada etapa de la cascada són complexos i estan construïts a base de classificadors més simples fent servir una de les quatre tècniques de reforçat (per ponderació

de vots): *Discrete Adaboost, Real Adaboost, Gentle Adaboost* o *Logitboost*. Amb el reforçat es crea un classificador fort basat en un (gran) conjunt de classificadors dèbils reponderant els exemples d'entrenament. El conjunt de classificadors dèbils es basen en una característica del conjunt de característiques en combinació d'una decisió binària basada en un valor llindar. Així, l'aprenentatge està basat en N exemples d'entrenament $(x_1, y_1), ..., (x_N, y_N)$ amb $x \in \mathbb{R}^k$ i $y_i \in \{-1, 1\}$ on x_i és un vector de K components. Cada component codifica una característica rellevant per la tasca d'aprenentatge on x_i és una característica de tipus Haar. La sortida de tipus +1 o -1 indica si el patró d'entrada conté una instància completa de l'objecte d'interès.

A continuació es mostra l'algorisme *Gentle Adaboost*, que es el que la llibreria OpenCV fa servir per defecte per entrenar els classificadors i el que s'ha triat per crear els classificadors d'aquest projecte:

Algorithm 6.1 Algorisme d'entrenament *Gentle Adaboost*

1. Donats N exemples $(x_1, y_1), ..., (x_N, y_N)$ amb $x \in \mathbb{R}^k, y_i \in \{-1, 1\}$
2. Començar amb pesos $w_i = 1/N, i = 1, ..., N$.
3. Repetir per $m = 1, ..., M$
   (a) Satisfer la funció de regressió $f_m(x)$ pel mètode de quadrats mínims de y_i a x_i amb pesos w_i
   (b) Fixar el valor $w_i \leftarrow w_i \cdot exp(-y_i \cdot f_m(x_i))$, $i = 1, ..., N$, i renormalitzar els pesos perquè $\sum_i w_i = 1$
4. Donar com a resultat el classificador $sign\left[\sum_{m=1}^{M} f_m(x)\right]$

Capítol 7

Implementació

L'aplicatiu que s'ha desenvolupat per aquest PFC funciona en GNU/Linux, havent-se desenvolupat íntegrament en un ordinador amb Ubuntu. La implementació del codi s'ha fet bàsicament en Python fent crides a funcions i mètodes de la llibreria OpenCV (implementada en C/C++ però amb interfície cap a Python), però algunes parts s'han escrit en C per qüestions d'eficiència. Python i OpenCV també estan disponibles per a Windows i Mac, pel que és viable portar el programa a aquestes altres plataformes si es modifica el codi amb el que es llencen events de teclat al sistema operatiu.

7.1 Entorn de desenvolupament

El projecte s'ha desenvolupat amb el sistema operatiu GNU/Linux Ubuntu en la seva versió 8.04 LTS. La programació s'ha portat a terme amb l'Eclipse Galileo amb el *plugin* PyDev. El control de versions del codi s'ha mantingut amb un servidor CVS instal·lat en el mateix ordinador amb el que s'ha estat programant, fent copies de seguretat manuals en un llapis USB i en un compte de correu electrònic. Pel desenvolupament de la interfície gràfica s'ha fet servir el Glade Interface Designer en la seva versió 3.4.5, aplicació per dissenyar interfícies per GTK+ i GNOME.

7.2 Instruccions bàsiques del funcionament del programa

El programa es pot executar tant per línia de comandes com mitjançant una senzilla interfície gràfica d'usuari. Per rebre instruccions sobre les opcions d'execució del programa per línia de comandes cal executar:

```
./gestrec.py -h
```

Aquesta sintaxi és d'ús comú en qualsevol programa de Linux. El programa respon mostrant una ajuda (en anglès) explicant les opcions d'execució del programa. Quan el programa s'executa sense indicar opcions entra automàticament en la fase de modelat del color de la pell. Si en canvi l'usuari indica el model de color a fer servir mitjançant la opció `-c` (o `--colorModel`) el programa passa directament a la fase de reconeixement de gestos.

D'altra banda, si l'usuari prefereix fer servir la interfície gràfica, més senzilla de fer servir, només hauria de fer doble clic sobre la icona del fitxer `gestrec_gui.py` que és executable i mostrarà directament la finestra "Principal".

7.3 Principals llibreries emprades

7.3.1 OpenCV

OpenCV és una llibreria de visió per computador implementada en C/C++ i amb interfície cap a altres llenguatges de programació, com ara l'Octave i el Python. Està publicada sota llicència BSD, i per tant l'ús acadèmic i comercial és lliure. La llibreria té més de 500 algorismes optimitzats i es fa servir arreu del món per a usos diversos tals com: l'art interactiu, la inspecció de mines, la construcció de mapes a partir d'imatges o en aplicacions de robòtica avançada. A hores d'ara la seva darrera versió estable és la 2.1, però el programa desenvolupat en per aquest PFC ha fet servir la 2.0. La llibreria està organitzada en els següents mòduls:

`cxcore` Hi trobem les estructures de dades bàsiques, l'àlgebra de matrius, transformacions de dades, persistència d'objectes, gestió de la memòria, gestió d'errors, càrrega dinàmica de codi, dibuixat i matemàtiques bàsiques.

`cv` Conté funcions de processament d'imatges, d'anàlisi d'estructura de la imatge, de moviment i seguiment d'objectes, reconeixement de patrons i calibració de la càmera.

`ml` Trobem algorismes d'aprenentatge *(Machine Learning)*, classificació i funcions d'anàlisi de dades.

`highgui` S'implementa la interfície gràfica d'usuari i l'emmagatzematge i la reproducció de vídeo.

`cvaux` Conté funcions experimentals auxiliars.

S'ha triat aquesta llibreria perquè la seva interfície en Python facilita la tasca del programador perquè al ser un llenguatge de programació de més alt nivell que C++ o C un es pot abstreure de problemes com la creació/destrucció d'estructures de dades i centrar-se així més en les tasques de visió per computador. Això no vol dir que el desenvolupador s'hagi d'oblidar què hi ha a sota del seu codi desenvolupat en Python; hi ha tasques que val més implementar-les amb C per qüestions d'eficiència.

7.3.2 SWIG

Els algorismes que s'han implementat amb C s'han portat a Python gràcies a la llibreria SWIG. SWIG significa *Simplified Wrapper Interface Generator*, i és, doncs, una eina simple per crear interfícies des de llenguatges de programació de baix nivell a llenguatges d'alt nivell. La Figura 7.1 mostra l'estructura de SWIG:

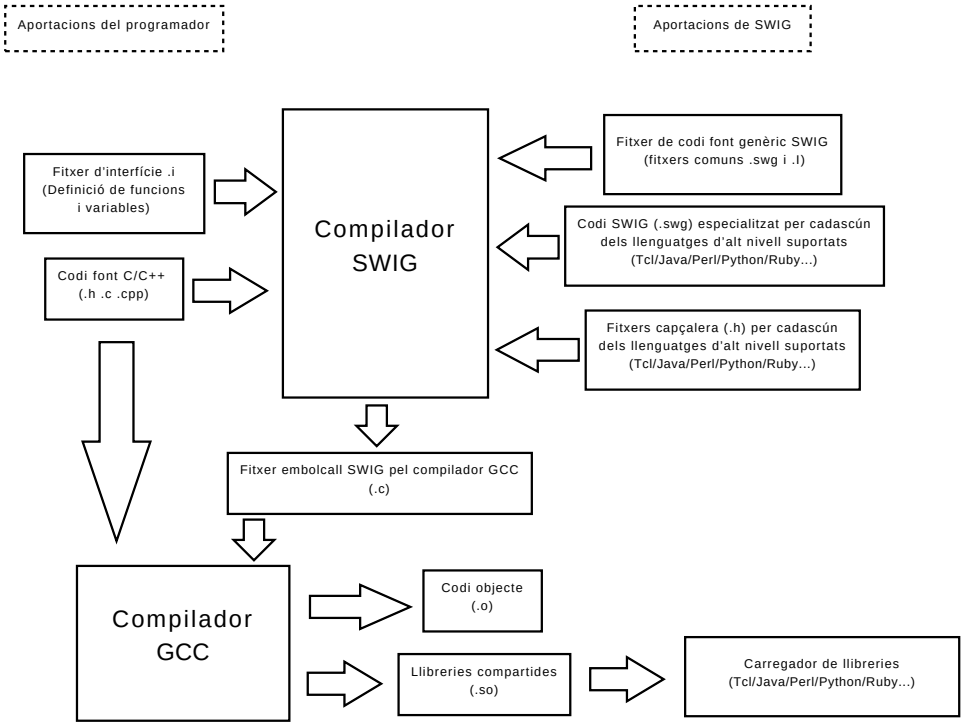


Figura 7.1: Funcionament de SWIG

Com a exemple il·lustratiu llistem aquí el codi i les passes que s'han seguit per portar les funcions de filtrat i modelat de color implementades en C a Python:

```
//Fitxer d'interfície color_model_swig.i
%module color_model_swig
%{
#include "Python.h"
#include "cxtypes.h"
#include "cv.h"
#include "cvaux.h"
%}
%inline %{
extern long max_occurrence;
extern int subcubeEdge;
extern double threshold;
%}
extern void learnColors (const CvMat* image, PyObject* colorMap);
extern void filterColors (const CvMat * image, CvMat * bwImage,
    PyObject* colorMap, int sgmType);
extern void normalizeColorModel(PyObject* colorMap);
```

En aquest fitxer es veu com els mètodes `learnColors`, `filterColors`, `normalizeColorModel`, i les variables `max_occurrence`, `subcubeEdge` i `threshold`

es publiquen per ser accedides des de Python. Per crear la interfície Python s'ha d'executar la següent comanda:

```
swig -python bgfg_codebook_swig.i
```

Amb la que obtindrem el fitxer `bgfg_codebook_swig_wrap.c` i el mòdul Python `bgfg_codebook_swig.py`. Un cop obtinguts aquest fitxers només cal compilar el codi C i obtenir la llibreria compartida `_bgfg_codebook_swig.so` que contindrà el codi compilat que s'executarà a les crides des de Python.

L'execució de les comandes necessàries per crear i compilar les interfícies SWIG es pot fer automàticament gràcies als Makefile's.

7.3.3 Psyco

Psyco és un mòdul d'extensió de Python que fa més ràpida l'execució dels programes i que no requereix fer canvis en el codi. És una mena de compilador JIT (*Just in Time compiler*) que emet codi màquina en blocs "al vol" de manera que l'interpret Python executa aquests blocs en comptes del llenguatge intermedi destinat a la màquina virtual. Els detalls sobre el sistema que fa servir es poden trobar al *paper*: "*Representation-based just-in-time specialization and the psyco prototype for python*" d'Armin Rigo (2004) [11]. Per fer-ho servir no cal res més que importar-ho i executar-ho al mòdul principal de qualsevol programa Python de la següent manera:

```
import psyco
psyco.full()
```

7.3.4 pickle

El mòdul `pickle` implementa un fonamental però potent algorisme per serialitzar i de-serialitzar objectes Python. S'ha fet servir per guardar a disc els models de color de pell de la següent manera:

```
#serialització
file = open(filename, "w")
pickle.dump(colorModel, file)
#de-serialització
file = open(filename,"r")
colorModel = pickle.load(file)
```

S'ha de remarcar que a Python tot és un objecte i en aquest cas `colorModel` és un objecte de tipus `dict` (diccionari) que guarda el model de color.

7.3.5 xtest

`xtest` és una interfície Python de la *X Protocol Test Suite* (joc de proves del protocol X) del sistema de finestres *X Window System*, que es fa servir a quasi totes les distribucions Linux. Aquesta interfície és la que s'ha fet servir per disparar els events de teclat de manera transparent pel sistema operatiu, que "creu" que han sigut provocats des del teclat. Per per posar un exemple, i gràcies a `xtest`, amb aquest parell d'instruccions és possible enganyar al sistema operatiu perquè cregui que s'ha pitjat "Control-Alt-Delete":

```
tester = XTest()
tester.fakeKeyEvent('Control-Alt-Delete')
```

7.3.6 PyGTK

PyGTK permet crear interfícies gràfiques d'usuari fent servir el llenguatge de programació Python. Fa servir la llibreria GTK+ (escrita en C) per oferir un exhaustiu conjunt de elements gràfics i altres facilitats de programació. El sistema d'escriptori GNOME, que ve per defecte amb moltes distribucions Linux, s'ha implementat amb GTK+ i moltes aplicacions de les que porta estan també implementades des de Python.

Amb el Glade Interface Designer i el PyGTK és possible crear interfícies gràfiques d'una manera senzilla i amb poques línies de codi.

7.3.7 yaml

YAML són les sigles de *Yet Another Marking Language;* és doncs un altre llenguatge de marcat més, com ho és l'XML. S'ha fet servir aquest llenguatge de marcat per desar i recuperar els perfils de configuració del programa, ja que aquest llenguatge de marcat és totalment intuïtiu i la llibreria yaml per Python permet manipular les estructures que es representen en YAML de forma senzilla. Així per exemple un fitxer de perfil de configuració del programa té aquesta forma:

```
colorModel: ./settings/colorModels/defaultColorModel.pck
gestures:
- classifier: ./settings/classifiers/gunLeftHaar.xml
  reversedKeyEvent: Right
  straightKeyEvent: Left
- classifier: ./settings/classifiers/palm.xml
  reversedKeyEvent: Escape
  straightKeyEvent: F5
```

Com es veu aquest format de fitxer és molt més llegible que l'XML. En ell es representa un diccionari amb dos claus: `colorModel` i `gestures`. La clau `colorModel` té com a valor la ruta d'un fitxer. La clau `gestures` té com a valor una llista i cada element de la llista és alhora un diccionari amb tres claus: `classifier`, `reversedKeyEvent` i `straightKeyEvent`. Aquesta estructura es manipula fàcilment d'aquesta manera:

```
#Per desar un diccionari a un fitxer
yaml.dump(dataMap, file)
#Per recuperar el diccionari
dataMap = yaml.load(file)
```

Abans de cridar a `yaml.dump` es crea i s'omple el diccionari `dataMap` (variable de tipus `dict`). Per tornar a recuperar l'estructura en un diccionari es crida a `yaml.load`.

7.4 Estructura del projecte

```
src/
└── grmodules/
    └── cvtasks/
        ├── bgfgsegmentation/
        │   ├── __init__.py
        │   ├── bgfg_codebook_swig.c
        │   ├── bgfg_codebook_swig.i
        │   └── Makefile
        ├── colormodel/
        │   ├── __init__.py
        │   ├── color_model_swig.c
        │   ├── color_model_swig.i
        │   └── Makefile
        ├── objectsdetection/
        │   ├── __init__.py
        │   ├── facedetect.py
        │   └── gestdetect.py
        ├── __init__.py
        ├── colorAcquisition.py
        └── gestureRecognition.py
```

```
...
src/
    grmodules/
        utils/
            __init__.py
            keyevents.py
            serialization.py
        __init__.py
    gui/
        windows
            gestrec.png
            main.glade
            Makefile
            new_gesture.glade
            settings.glade
        __init__.py
        gestrec_cli.py
        main.py
        new_gesture.py
        settings.py
    settings/
        classifiers/
            gunLeftHaar.xml
            haarcascade_frontalface.xml
            ...
        colorModels/
            defaultColorModel.pck
        profiles/
            default.yaml
    config.py
    gestrec_gui.py
    gestrec.py
    Makefile
    shared.py
```

- `grmodules/`

  Aquesta carpeta conté els fitxers on estan implementades totes les funcions útils per al programa.

- `grmodules/cvtasks/`

  En aquesta carpeta trobem allò necessari per fer les diferents tasques de visió per computador i conté els fitxers:

  - `colorAcquisition.py`: implementació en Python de l'obtenció del model de color de la pell (punt 4.2).
  - `gestureRecognition.py`: implementació en Python de la detecció de gestos (punt 4.1).

- `grmodules/cvtasks/bgfgsegmentation/`

  Aquesta carpeta conté els fitxers que implementen les funcions necessàries per fer la segmentació del fons, que són:

  - `bgfg_codebook_swig.c`: implementació en C de les funcions de segmentat de fons.
  - `bgfg_codebook_swig.i`: fitxer d'interfície SWIG per fer possible la crida de les funcions implementades en C des de Python.
  - `Makefile`: fitxer make per compilar el codi en C i crear les interfícies Python mitjançant SWIG.

- `grmodules/cvtasks/colormodel/`

  Aquesta carpeta conté els fitxers que implementen les funcions necessàries per construir el model de color de la pell i fer la segmentació per color, que són:

  - `bgfg_model_swig.c`: implementació en C de les funcions de construcció del model de color i segmentat.
  - `bgfg_codebook_swig.i`: fitxer d'interfície SWIG per fer possible la crida de les funcions implementades en C des de Python.
  - `Makefile`: fitxer make per compilar el codi en C i crear les interfícies Python mitjançant SWIG.

- `grmodules/cvtasks/objectsdetection/`

  Aquesta carpeta conté els fitxers que implementen les funcions necessàries pel reconeixement de cares i gestos, que són:

  - `facedetect.py`: implementació en Python de les funcions necessàries per la detecció de la cara.
  - `gestdetect.py`: implementació en Python de les funcions necessàries per la detecció dels gestos.

- `grmodules/utils/`

  Aquesta carpeta conté els fitxers que implementen algunes funcions útils amb:

  - `keyevents.py`: implementació en Python de les funcions necessàries per disparar events de teclat.
  - `gestdetect.py`: implementació en Python de les funcions necessàries per la serialització d'objectes (útils per guardar el model de color de la pell).

- `gui/`

  Aquesta carpeta conté tot allò necessari per la interfície gràfica d'usuari:

  - `gestrec_cli.py`: fitxer necessari per invocar al programa principal des de la interfície gràfica.
  - `main.py`: conté el codi necessari per visualitzar i controlar la finestra "Principal".
  - `new_gesture.py`: conté el codi necessari per visualitzar i controlar la finestra "Nou gest".
  - `settings.py`: conté el codi necessari per visualitzar i controlar la finestra "Modificar configuració".

- `gui/windows/`

  Aquesta carpeta conté els fitxers que modelen la interfície gràfica d'usuari:

  - `gestrec.png`: icona del programa.

  - `main.glade:` model de la finestra "Principal" creat amb Glade.
  - `Makefile:` fitxer make per poder convertir els fitxers Glade a fitxers XML.
  - `new_gesture.glade:` model de la finestra "Nou gest" creat amb Glade.
  - `settings.glade:` model de la finestra "Modificar configuració" creat amb Glade.

- `settings/`

  En aquesta carpeta es guarden els diferents perfils de configuració del programa i fitxers relacionats.

- `settings/classifiers/`

  En aquesta carpeta es guarden els classificadors basats en característiques Haar en fitxers .xml

- `settings/colorModels/`

  En aquesta carpeta es guarden els models de color de la pell en fitxers .pck (pickle).

- `settings/profiles/`

  En aquesta carpeta es guarden els perfils de configuració en fitxers .yaml (Yet Another Marking Language).

- `config.py`

  Aquest mòdul conté les diferents variables de configuració del programa alguns valors de les quals es fixen en carregar un perfil de configuració i són consultades arreu d'aquest.

- `gestrec_gui.py`

  Mòdul principal de la interfície gràfica d'usuari.

- `gestrec.py`

  Mòdul principal del programa.

- `Makefile`

  Fitxer make que invoca tots els altres fitxers make del projecte per construir i compilar allò necessari per l'execució d'aquest.

- `shared.py`

  Aquest mòdul conté variables que són compartides per diferents mòduls del programa.

7.5 Implementació de les funcionalitats bàsiques

7.5.1 Aprenentatge del fons

Com ja s'ha explicat per construir el model de color de la pell es capturen els colors de la mà de l'usuari, que apareix aïllada a la imatge un cop s'ha segmentat aquesta gràcies a la sostracció del fons. Per això es necessari modelar primer el fons sobre el que l'usuari hi posarà la mà, així que quan es demana al programa que construeixi el model de color de la pell es mostra una finestra amb la visió de la càmera en temps real i es marca sobre aquesta l'àrea del quadrant superior dret, d'on el programa n'aprendrà el fons executant, entre altres coses, aquestes instruccions per a cada imatge capturada per la càmera:

```
cvCvtColor(frame, frame, config.RGB2X)
colorCaptureArea = cvCloneImage(cvGetSubRect(frame, rect))
bgCodeBookUpdate(colorCaptureArea, codebook)
cvCvtColor(frame, frame, config.X2RGB)
```

El fitxer config.py conté variables que permeten al programador modificar de forma centralitzada el comportament de les diferents tasques de visió per computador

1. Traslladem la imatge que ha capturat la càmera de l'espai de color RGB a un altre espai apte per la tècnica que s'està emprant (s'expliquen més detalls sobre això al punt 6.1). Per defecte el valor de la variable `config.RGB2X` és `CV_RGB2Lab`, valor que es fa servir en la llibreria OpenCV per traslladar els colors d'una imatge de RGB a CIELab.

2. Clonem el subrectangle corresponent al quadrant superior dret de la imatge i el guardem a `colorCaptureArea`.

3. Actualitzem el *codebook* perquè aprengui els colors del `colorCaptureArea`.

4. Tornem a traslladar els colors de la imatge al RGB. Per defecte el valor de la variable `config.X2RGB` és `CV_Lab2RGB`.

Un cop s'han analitzat prou imatges (`config.nFramesToLearnBackground`) s'executa la següent instrucció per fer neteja dels valors residuals del *codebook*:

```
bgCodeBookClearStale(codebook)
```

Cal remarcar que de la mateixa manera que s'invoquen les funcions implementades en C de la llibreria OpenCV des de Python, aprofitant així la rapidesa del codi C compilat i al mateix temps la versatilitat del codi Python interpretat, s'han implementat també en C les funcions útils per la segmentació primer pla/fons i s'ha fet servir la llibreria SWIG per embolcar-les perquè es comportin com qualsevol altre funció implementada en Python.

7.5.2 Modelat del color de la pell

Un cop el programa disposa d'un *codebook* que modela el fons, l'usuari ha de moure la mà per la mateixa àrea d'abans perquè el programa se'n faci un model del color de la pell (veure els detalls al punt 6.2). El programa construeix el model del color de la pell de l'usuari executant, entre altres, les instruccions:

```
cvCvtColor(frame, frame, config.RGB2X)
colorCaptureArea = cvCloneImage(cvGetSubRect(frame, rect))
segmentFG(colorCaptureArea, codebook, iMaskCodeBook)
cvZero(tmpImg8U3ChCCA)
cvCopy(colorCaptureArea, tmpImg8U3ChCCA, iMaskCodeBook)
learnColors(tmpImg8U3ChCCA, shared.colorModel)
```

1. Traslladem la imatge que ha capturat la càmera de l'espai de color RGB a l'espai de color CIELab.

2. Clonem el subrectangle corresponent al quadrant superior dret de la imatge i el guardem a `colorCaptureArea`.

3. Fem un segmentat de fons obtenint una màscara binària `iMaskCodeBook` que indica quins píxels corresponen a allò que està en primer pla, és a dir, la mà de l'usuari que ha aparegut per sobre del fons que modela el `codebook`.

4. Posem a 0 els valors d'una imatge temporal (`tmpImg8U3ChCCA`) de 8 bits i tres canals que té la mateixa mida que `colorCaptureArea`.

5. Copiem a `tmpImg8U3ChCCA` tots els píxels de `colorCaptureArea` que estan actius a la màscara `iMaskCodeBook` obtenint així una imatge on només hi apareix la mà de l'usuari.

6. Aprenem els colors de la pell de l'usuari que apareixen a la imatge `tmpImg8U3ChCCA` actualitzant els acumuladors de l'histograma corresponent als subcubs de l'espai CIELab on han "caigut" els valors de color de cada píxel.

Aquí de nou es crida a mètodes (segmentFG i learnColors) implementats en C des del codi Python.

Val la pena endinsar-se aquí una mica en el codi que hi ha sota el mètode `learnColors` per entendre perquè està implementat en C:

```
void learnColors (const CvMat * image, PyObject* colorMap){
   int x, y;
   for(y = 0; y < image->rows; y++){
      const uchar* p = image->data.ptr + image->step*y;
      for( x = 0; x < image->cols; x++, p += 3){
         insertColor(p, colorMap);
      }
   }
}
void insertColor(const uchar p[], PyObject* colorMap){
   PyObject* pyValue;
   long value = -1;
   if(!(p[0]==0 && p[1]==0 && p[2]==0)){
      PyObject* tuple = PyTuple_New(3);
      //Create a Tuple with three three representing
      //the origin of the subcube
      PyTuple_SetItem(tuple,0,PyInt_FromLong(p[0]/subcubeEdge));
      PyTuple_SetItem(tuple,1,PyInt_FromLong(p[1]/subcubeEdge));
      PyTuple_SetItem(tuple,2,PyInt_FromLong(p[2]/subcubeEdge));
      //Check if the subcube is already in the dictionary
      pyValue = PyDict_GetItem(colorMap, tuple);
      if(pyValue != NULL){
         //Increment the subcube hit counter
         value = PyInt_AS_LONG(pyValue) + 1;
         //Mantain the current maximum
         if(value > max_occurrence) max_occurrence = value;
         PyDict_SetItem(colorMap, tuple, PyInt_FromLong(value));
         Py_DECREF(pyValue);
      }else{
         //First insertion into the dictionary
         PyDict_SetItem(colorMap, tuple, PyInt_FromLong(1));
      }
         Py_DECREF(tuple); } }
```

Aquí veiem com al mètode `learnColors` es fa un recorregut seqüencial de la imatge enviant al mètode `insertColor` el vector p, que conté els valors dels tres canals d'un píxel. En aquest mètode, si el vector p no correspon a un píxel amb valors a 0, es construeix la tupla $\left(\left\lfloor \frac{p[0]}{subcubeEdge} \right\rfloor, \left\lfloor \frac{p[1]}{subcubeEdge} \right\rfloor, \left\lfloor \frac{p[2]}{subcubeEdge} \right\rfloor\right)$ per actualitzar els comptadors corresponents al diccionari `colorMap` a cada ocurrència. A més es manté el valor de la variable `max_ocurrence`, que indica el valor màxim de l'histograma i que serà útil per normalitzar-ho.
Queda clar aquí com un codi que treballa a tant baix nivell (recorrent els bytes de la imatge) val la pena implementar-ho amb C perquè vagi ràpid. Es veu també com en aquest cas és necessari decrementar a mà els comptadors a les referències d'objectes Python (crides a Py_DECREF) perquè el garbage collector vagi alliberant memòria, tasca que quan es treballa només amb Python és transparent al programador.
Un cop s'ha omplert l'histograma amb una àmplia gama de colors de la pell aquest s'ha de normalitzar per encabir tots els comptadors dins del rang [0...1]. Per tal de saber quin és el moment en el que el programa ja ha modelat els colors de la pell de l'usuari s'indica en pantalla quants subcubs de l'espai CIELab han estat "tocats". Quan l'usuari estigui movent la mà per l'àrea de captura de colors s'adonarà de que arriba un moment en que el comptador de subcubs "tocats" gairebé no avança, llavors pot pitjar la barra espaiadora per finalitzar el procés de modelat dels colors de la pell. Cal recordar que la tasca de construcció de models de color està reservada a usuaris experts que coneguin una mica el procés, no a usuaris bàsics. Un cop s'ha pitjat la barra espaiadora es normalitza l'histograma executant la següent instrucció:

```
normalizeColorModel(shared.colorModel)
```

Aquest mètode està, de nou, implementat en C de la següent manera:

```
void normalizeColorModel(PyObject* colorMap){
   PyObject *key, *value;
   Py_ssize_t pos = 0;
   double ratio;
   //Divide each value in the dictionary by the max_occurrence
   while (PyDict_Next(colorMap, &pos, &key, &value)) {
      ratio = PyInt_AS_LONG(value) / (double)max_occurrence;
      PyObject *o = PyFloat_FromDouble(ratio);
      PyDict_SetItem(colorMap, key, o);
      Py_DECREF(o);
   }
}
```

Aquí simplement es divideixen tots els valors de l'histograma entre el valor màxim `max_ocurrence`.

7.5.3 Refinat del model de color de la pell

Un cop s'ha finalitzat el procés de construcció del model del color de la pell i s'estigui ja fent el segmentat per color, l'usuari el pot refinar i tornar a desar-ho prement un altre cop la barra espaiadora. Així, per esborrar colors de l'histograma, com per exemple blancs de la paret de fons que s'hagin incorporat al model per error, es pot fer clic amb el botó esquerre sobre les zones on hi apareguin. S'ha implementat un mètode que captura aquests events de ratolí i que analitza els valors dels píxels al voltant del punt de la imatge que s'ha clicat per acte seguit esborrar-los del diccionari que guarda el model de color.

Aquesta i altres funcionalitats addicionals es comuniquen a l'usuari en una petita ajuda en línia que es mostra en executar el programa des de línia de comandes i estan documentades de forma una mica més extensa a la petita ajuda que es pot invocar executant el programa amb el modificador -h

7.5.4 Segmentat per color

Quan l'usuari prem la barra espaiadora per finalitzar el modelat del color de la pell el programa entra en la fase de reconeixement de gestos. El primer que es fa en aquesta fase és el segmentat del color de la pell. Així, per a cada imatge capturada per la càmera s'executa la instrucció:

```
color_model_swig.filterColors(frame, tmpImg8U1Ch, shared.colorModel,
    sgmType)
```

Aquest mètode rep la imatge capturada per la càmera `(frame)` un cop ja ha estat traslladada a l'espai CIELab, una imatge d'un canal o màscara (`tmpImg8U1Ch`) amb tots els seus valors a 0 on es marcaran els píxels corresponents a la pell, el model de color `shared.colorModel` i la variable `sgmType` que indica si es vol aplicar operacions addicionals sobre la màscara binària abans de finalitzar el mètode (el sentit d'aquest paràmetre s'explicarà en breu). Un cop més el mètode, com que recorre seqüencialment tots els píxels de la imatge, està implementat en C de la següent manera:

```
void filterColors(const CvMat * image, CvMat * bwImage, PyObject*
    colorMap, int sgmType){
   int x, y;
   double value;
   PyObject* pyValue;
   PyObject* tuple = PyTuple_New(3);
   for( y = 0; y < image->rows; y++ ){
      uchar* p = image->data.ptr + image->step*y;
      uchar* bw_p = bwImage->data.ptr + bwImage->step*y;
      for( x = 0; x < image->cols; x++, p += 3, bw_p += 1){
         //Which subcube hits this color?
         PyTuple_SetItem(tuple, 0, PyInt_FromLong(p[0]/subcubeEdge));
         PyTuple_SetItem(tuple, 1, PyInt_FromLong(p[1]/subcubeEdge));
         PyTuple_SetItem(tuple, 2, PyInt_FromLong(p[2]/subcubeEdge));
         //Check if the subcube appears in the dictionary (histogram)
         pyValue = PyDict_GetItem(colorMap, tuple);
         if(pyValue!=NULL){
            value = PyFloat_AS_DOUBLE(pyValue);
            //Check if its value is greater than the threshold
            if(value > threshold){
               //This is a skin color
               bw_p[0]=255;
            }
         }
      }
   }
   //Apply some operations to the mask (bwImage)
   //in order to clean spare little areas?
   switch(sgmType){
      case 0:
         cvSegmentFGMask(bwImage, 1, 4.f, 0, cvPoint(0,0));
      case 1:
         cvErode(bwImage, bwImage, NULL, 1);
         cvDilate(bwImage, bwImage, NULL, 1);
         cvErode(bwImage, bwImage, NULL, 1);
         cvDilate(bwImage, bwImage, NULL, 2);
      default:
         ;
   }
}
```

En aquest codi veiem com es recorre la imatge píxel a píxel i es mira si el color d'aquests "cau" dins d'uns dels subcubs que conté el model de color. Si el triplet corresponent a l'origen de coordenades del subcub de color on ha "caigut" el píxel està a l'histograma amb un valor per sobre del llindar (`threshold`) llavors es marca el píxel a la màscara binària com a pell. Després d'això ja tenim marcats a `bwImage` els píxels que es consideren pell. En aquest moment entra en acció la variable `sgmType;` segons el seu valor la

màscara binària es deixarà tal com està (`default`), se li aplicarà la operació `cvSegmentFGMask` o se li aplicaran operacions d'erosió i dilatat. El mètode `cvSegmentFGMask` es fa servir internament a la implementació de segmentat de fons que fa l'OpenCV i és útil per millorar la màscara binaria netejant les petites regions disperses que puguin haver quedat. Les operacions d'erosió i dilatat busquen el mateix objectiu. S'ha trobat experimentalment, però, que no es necessari treure aquest soroll de la màscara binària i és per això que la opció per defecte és no fer res.

L'usuari pot alternar entre les diferents opcions de segmentat (sgmType) pitjant la tecla c. De nou aquesta funcionalitat addicional està citada a l'ajuda del programa.

7.5.5 Reconeixement facial i selecció del tronc

Per detectar la cara de l'usuari es fa servir un classificador en cascada de característiques Haar. Un cop s'ha detectat la cara es retalla l'àrea que queda per sota d'aquesta per buscar-hi més tard els gestos. Per obtenir el rectangle que emmarca la cara de l'usuari es fa servir la instrucció:

```
faceRectsList = facedetect.detect(smallFrame)
```

`facedetect.detect` crida internament a `cvHaarDetectObjects` funció d'OpenCV que entre d'altres paràmetres rep la imatge on cercar la cara i el classificador en cascada que detecta cares frontals, retornant una llista de rectangles corresponents a les àrees on s'han trobat cares. D'aquesta manera obtenim una llista de rectangles del que només es fa servir el primer, doncs a l'escena només hi ha d'haver-hi el ponent. Acte seguit es retalla l'àrea `rectBelowFace` que es correspon al tronc de l'usuari:

```
imgRectBelowFace = cvGetSubRect(filteredImg,rectBelowFace)
```

7.5.6 Reconeixement de gestos

Els gestos es reconeixen de la mateixa manera que les cares: amb classificadors Haar. En aquest cas però es crida a la instrucció:

```
gestId, flag = gestdetect.detect(imgRectBelowFace)
```

Aquesta crida retorna una tupla amb dos valors: l'identificador del gest que s'ha detectat a la imatge `imgRectBelowFace` (si és que se n'ha detectat cap)

i un indicador (`flag`) que diu si ha calgut girar la imatge per trobar-hi un gest.

7.6 Entrenament dels classificadors

Per entrenar un classificador per un cert gest primer cal tenir una àmplia base d'imatges on només hi aparegui el gest. Per aconseguir aquestes imatges s'han gravat diversos vídeos fent el gest en diferents condicions de llum i després s'ha retallat de forma manual, imatge a imatge, el gest en qüestió amb l'ajuda d'un programa dissenyat a tal efecte: `imageclipper`[1]

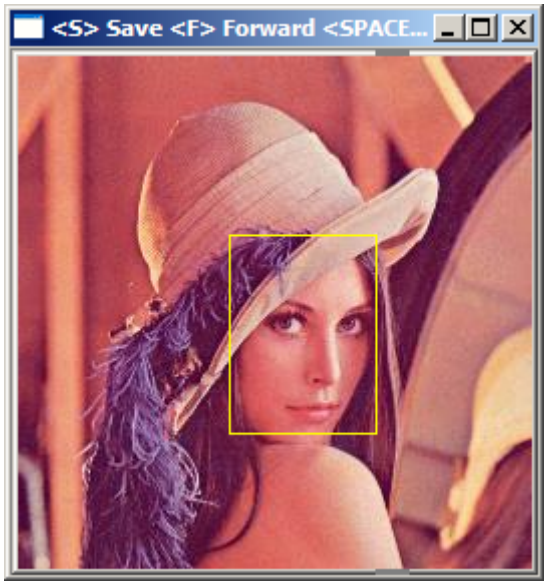


Figura 7.2: imageclipper

`imageclipper` és una aplicació que permet obrir totes les imatges d'un directori o mostrar un vídeo imatge a imatge per retallar-hi manualment les àrees d'interès (veure Figura 7.2). Un cop es selecciona l'àrea amb el ratolí es pot guardar a disc prement la barra espaiadora o passar a la següent imatge sense guardar res pitjant la tecla F. Acte seguit es mostra la següent imatge amb el requadre a la mateixa posició. El requadre es pot moure i redimensionar fàcilment amb el teclat, fet que permet retallar massivament imatges, una rere l'altre, sense haver de fer servir casi el ratolí. Les imatges retallades

[1] `http://code.google.com/p/imageclipper` (darrera visita 09/06/10)

queden guardades per defecte en el subdirectori imageclipper i cada captura es guarda a un fitxer amb el següent format de nom:

```
%i.%e_%04x_%04y_%04w_%04h.png
On:
        %i - nom del fitxer origen sense extensió
        %e - extensió del fitxer origen
        %04x - coordenada X de l'origen del requadre amb 4 xifres
        %04y - coordenada Y de l'origen del requadre amb 4 xifres
        %04w - ample del requadre amb 4 xifres
        %04h - llarg del requadre amb 4 xifres
```

Un cop tenim totes les imatges retallades necessitem un fitxer de descripció que contingui el llistat dels noms dels fitxers d'imatges seguits del nombre d'àrees d'interès i les seves posicions i mides. En el nostre cas només tenim un àrea d'interès per imatge (doncs només guardem la imatge de la mà) que té l'origen a la posició 0 0 i té tot l'ample i llarg de la imatge. Per generar aquest fitxer de descripció executem les comandes:

```
$ cd <your working directory>
$ find imageclipper -name '*.png' -exec identify -format \
'%i 1 0 0 %w %h' \{\} \; > samples.dat
```

Amb les que:

1. Canviem al directori de treball (on estan els vídeos).

2. Cerquem totes els fitxers al subdirectori imageclipper amb extensió png i obtenim la informació del seu format amb la utilitat `identify` (de la llibreria ImageMagick[2]) construint per cadascun la cadena '%i 1 0 0 %w %h' on %i és el nom del fitxer, 1 és el nombre d'objectes a la imatge, 0 0 la posició origen del requadre que l'emmarca i %w %h l'ample i llarg de la imatge. S'escriuen aquestes cadenes, línia a línia, al fitxer de descripció `samples.dat`

A partir d'aquest fitxer de descripció creem el fitxer de mostres de 20x20 (`samples.vec`) per l'entrenament del classificador amb l'ajuda del programa `opencv_createsamples` executant la següent comanda:

```
$ opencv_createsamples -info samples.dat -vec samples.vec -w 20 -h 20
```

Abans d'executar l'aplicació d'entrenament de classificadors, cal també disposar d'una gran base d'imatges negatives on no hi apareguin els objectes

[2] `http://www.imagemagick.org` (darrera visita el 09/06/10)

pels quals es vol aconseguir el classificador. En aquest projecte es van fer servir les 3019 imatges trobades al repositori de codi del tutorial de haartraining escrit per Naotoshi Seo[3]. Un cop tenim el fitxer de mostres d'entrenament (`samples.vec`) i un fitxer de descripció (`negatives.dat`) amb tots els noms dels fitxers de les imatges negatives, executem l'aplicació d'entrenament de classificadors amb una comanda com aquesta:

```
$ opencv_haartraining -data haarcascade -vec samples.vec \
-bg negatives.dat -nstages 20 \
-nsplits 2 -minhitrate 0.999 -maxfalsealarm 0.5 \
-npos 500 -nneg 1000 -w 20 \
-h 20 -nonsym -mem 1024 -mode ALL
```

On:

- -data: haarcascade serà el subdirectori on s'aniran guardant els resultats de cada fase d'entrenament i el nom del classificador resultant al final del procés: haarcascade.xml
- -nstages: es posa un límit de 20 fases d'entrenament.
- -nsplits: l'arbre de decisió es subdivideix en 2 branques per node.
- -minhitrate: la proporció mínima d'encerts es fixa a 0.999, i per tant, havent-hi 20 nivells a l'arbre decisional, buscarem obtenir una taxa d'encerts total de $0.999^{20} \approx 0.98$
- -maxfalsealarm: la proporció màxima d'alarmes falses es fixa a 0.5, i per tant buscarem obtenir una proporció de falses alarmes total de $0.5^{20} \approx 9.6e-07$
- -npos: es faran servir 500 mostres positives.
- -nneg: es faran servir 1000 mostres negatives.
- -w -h: la mida de les mostres serà de 20x20, mida en la que segons Lienhart et. al[8] s'aconsegueix la millor taxa d'encerts en la detecció de cares, però amb la que s'han aconseguit resultats satisfactoris en la detecció de gestos.

[3]Tutorial OpenCV haartraining - per Naotoshi Seo:
`http://note.sonots.com/SciSoftware/haartraining.html` (darrera visita el 09/06/10)
Imatges negatives
`http://tutorial-haartraining.googlecode.com/svn/trunk/data/negatives/`

- -nonsym: els objectes a detectar no presenten simetria vertical, pel que es faran servir totes les característiques Haar, detectant així contrastos a esquerra i dreta dels *kernels*.
- -mem: límit de memòria a utilitzar de 1024 Mb.
- -mode ALL: es fa servir un conjunt estès de característiques Haar (veure Figura 6.3).

Capítol 8

Resultats

8.1 El programa en funcionament

8.1.1 Modelat del fons i del color de la pell

El programa es distribueix amb un model de color que funciona pel to de pell caucàsic, però un usuari avançat pot crear-ne un de nou. Quan demana fer un modelat del color de la pell el primer que es fa és un aprenentatge del fons, llavors se li mostra una finestra com la de la Figura 8.1. En ella es veu un àrea marcada les imatges de les quals es prendran com a fons. Sota aquesta àrea es mostra un comptador que va decreixent fins al zero, marcant el progrés del procés de modelat.

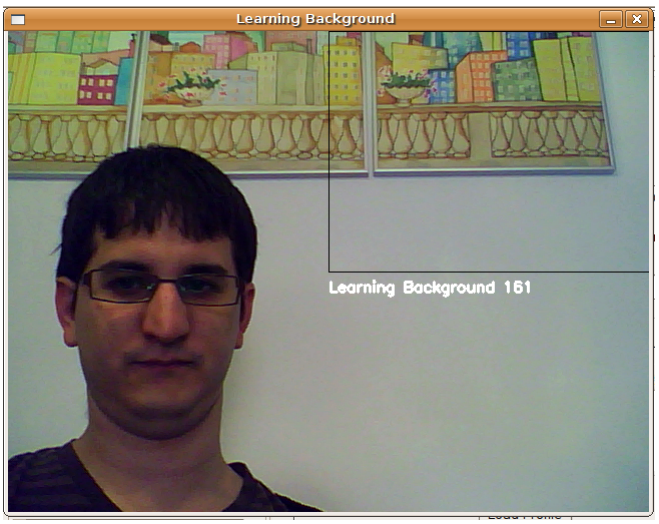


Figura 8.1: Finestra d'aprenentatge del fons

Un cop ha finalitzat el procés d'aprenentatge del fons es mostren les dos finestres que es veuen a la Figura 8.2.

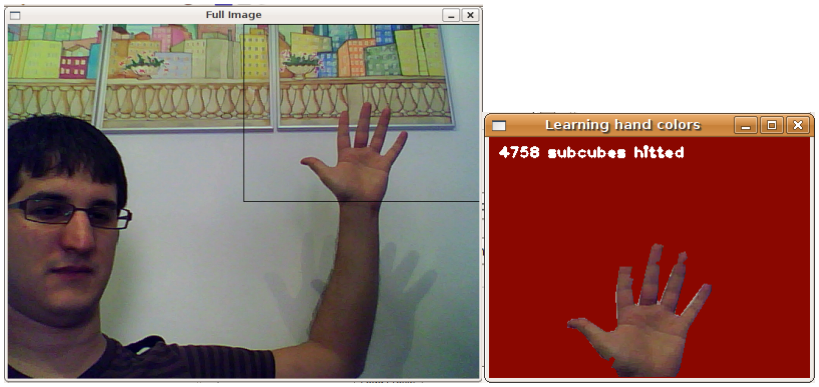


Figura 8.2: Finestres de modelat del color de la pell

L'usuari ara ha de moure la mà a la mateixa àrea marcada i la veurà aïllada del fons en una petita finestra que va comptant quants subcubs de l'espai de color han sigut "tocats", és a dir, en quants subcubs han caigut els colors de la mà analitzats. Quan el comptador "subcubes hitted" s'estabilitza l'usuari ha de prémer la barra espaiadora per finalitzar el procés i guardar el model de color.

8.1.2 Refinat del model de color

Quan el model de color està creat de nou conté alguns colors no-pell que s'han colat al model provinents del fons. L'usuari pot, però, jugar amb el nivell del valor llindar (amb les tecles +/-) perquè es filtrin aquells colors que estan al model en una proporció baixa, a més de tenir la possibilitat d'esborrar definitivament certs colors clicant amb el ratolí sobre les àrees on hi apareixen.
A continuació es mostren unes figures que il·lustren el procés de millora del model de color de la pell:

- A la Figura 8.3 veiem funcionant el filtratge de color amb el model de color de la pell creat en l'apartat anterior amb un valor llindar negatiu (per sota d'aquest llindar no quedarà cap valor de l'histograma), de manera que a la imatge es mostren tots els colors que figuren al model (incloent els que a l'histograma mostren una proporció molt baixa):

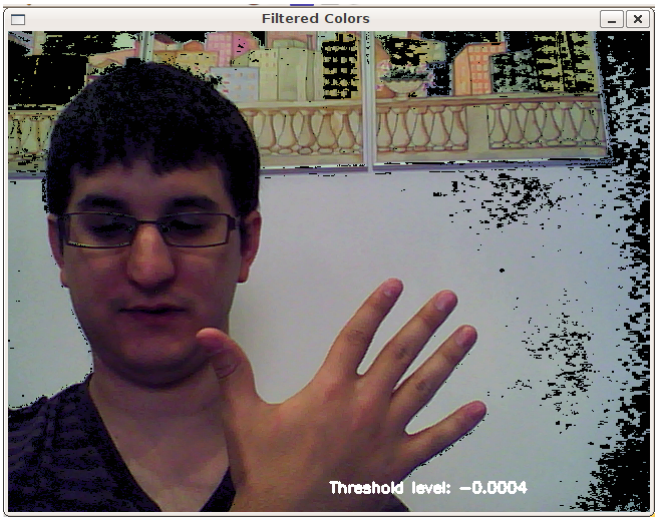


Figura 8.3: Filtrat de color sense llindar

Veiem com a la imatge no falta cap color de la pell, però sobren els colors de la paret, de la samarreta i alguns colors dels quadres, que figuren al model del color provinents del fons.

- A la Figura 8.4 veiem que si s'augmenta el valor del llindar llavors desapareixen de la imatge aquells conjunts de color que tenen un valor baix a l'histograma. No és convenient, però, pujar molt aquest llindar, doncs el model contindrà alguns conjunts de color de la pell en una proporció molt baixa que llavors farem desaparèixer de la imatge.

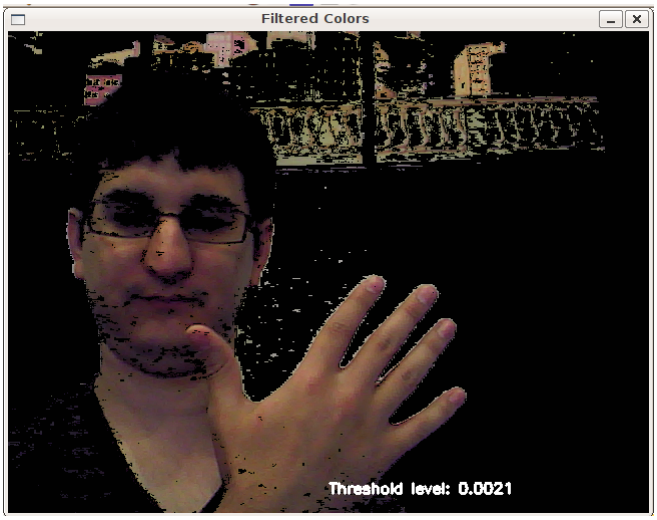


Figura 8.4: Filtrat de color pujant el llindar

A la imatge es veu com algunes àrees de la cara s'han esborrat, mentre es manté alguns trossets residuals de la paret blanca i la samarreta marró. Es pot considerar també que alguns colors del quadre sobren, però la majoria estan dins de la gamma del color de la pell.

- A la Figura 8.5 es mostra el resultat del filtrat amb un llindar baix, però a més s'han esborrat a mà els conjunts de colors de la paret i la samarreta.

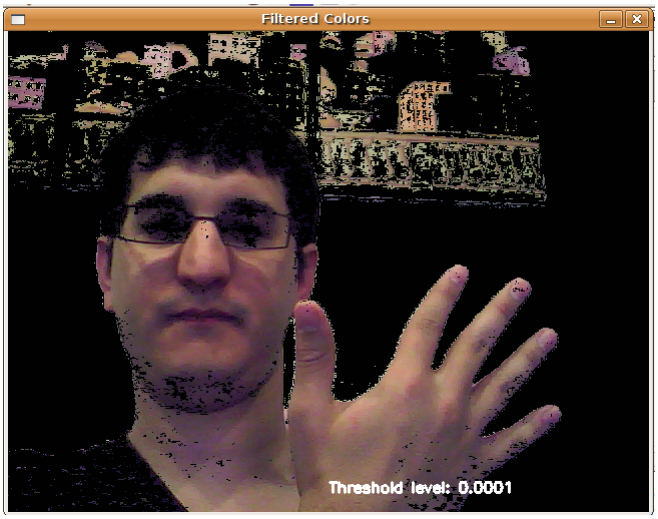


Figura 8.5: Filtrat amb model retocat

Aquesta és la millor manera d'aconseguir un bon model de color, cal jugar amb el valor del llindar i alhora esborrar colors del model amb uns quants clics. Quan l'usuari que està construint el model considera que és apropiat per ser guardat només ha de prémer la barra espaiadora per deixar-ho a disc.

8.1.3 Reconeixement de gestos

El mode de reconeixement de gestos està pensat per córrer en segon pla, ja que a l'usuari li interessa veure en pantalla les diapositives de la presentació; encara que es pot veure el programa en funcionament amb dos finestres: una que mostra en petit la imatge que capta la càmera, emmarcant-hi en vermell la cara de l'usuari i l'altre on apareixen les imatges a mida original però filtrant els colors de la pell. En aquesta darrera finestra es mostra també emmarcada en blau l'àrea per sota de la cara, àrea on el programa buscarà els gestos. Si hi detecta algun gest s'emmarcarà aquest en verd o vermell, depenent de si s'ha trobat del dret o del revés. La Figura 8.6 il·lustra aquesta explicació.

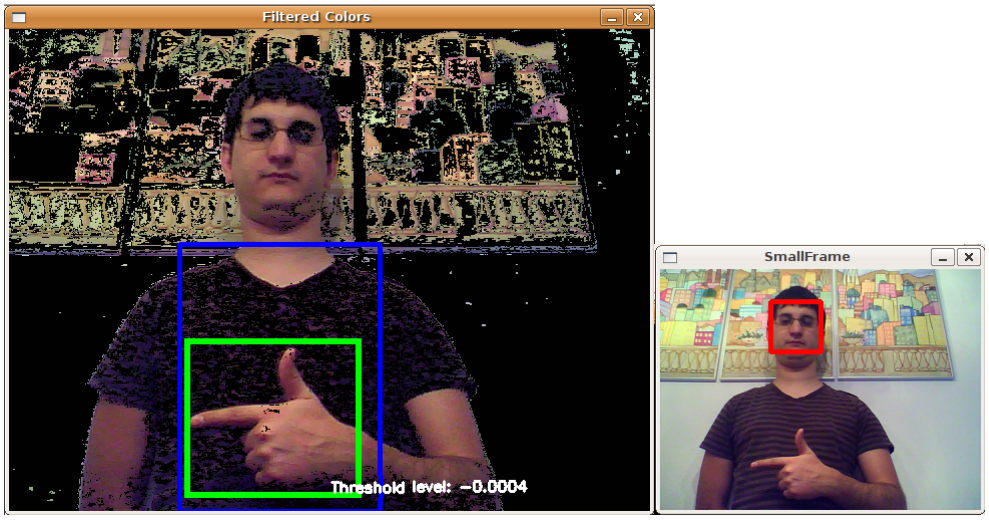


Figura 8.6: Finestres de reconeixement de gestos

8.2 Crítica sobre les tècniques emprades

El sistema de modelat i filtrat de colors de la pell funciona satisfactòriament, encara que tot és millorable. I una millora en aquest sentit inclouria un mètode adaptatiu que anés modificant el model de color així van canviant les condicions de la llum. El sistema emprat abasta, però, una gamma de colors àmplia, ja que captura els subespais de l'espai CIELab on cauen els colors de la pell, estiguin aquests dispersos o condensats. Altres sistemes es basen en el càlcul de la distància dels colors a un punt que es suposa el centre de la gamma de colors de la pell, de manera que es depèn de la tria

d'un espai de color on sempre apareguin junts, fins i tot amb canvis de llum. Això és una mica difícil, doncs el color de la pell canvia segons molts factors incontrolables, de manera que la concentració d'aquestos en un espai de color no sempre és una realitat. L'espai CIELab ens assegura, però, que els colors es disposen d'una forma uniforme perceptualment i nosaltres marquem els subcubs de l'espai on s'han trobat colors de la pell, així que no depenem de la seva dispersió o concentració, sinó que només hem de confiar en que a l'espai CIELab es trobin els colors semblants en posicions properes. La subdivisió de l'espai CIELab es fa per defecte en subcubs d'aresta 4, però aquest valor es pot modificar al fitxer `config.py` o invocant el programa des de línia de comandes amb l'opció adient.

El reconeixement de gestos mitjançant classificadors en cascada basats en característiques de tipus Haar ha sigut el principal llast d'aquest projecte. S'ha barallat molt per aconseguir uns pocs classificadors i se n'ha hagut de llançar molts per inservibles. El pitjor de tot és que després dels llargs entraments molts cops se n'obtenia un classificador que no funcionava, amb el problema de que és molt difícil analitzar el perquè. El més segur és que la nul·la familiarització que es tenia amb la tècnica en començar el projecte sigui la font de tants problemes, però s'ha trobat a faltar una eina de visualització del funcionament dels classificadors que permetés esbrinar què s'estava fent malament. D'existir una representació gràfica de les disposicions dels *kernels* de característiques Haar que el classificador fa servir el progrés en aquest àmbit hagués estat un altre. Potser per detectar els gestos, un cop aconseguit l'entorn controlat de que disposàvem, hagués estat més productiu implementar algun mètode basat en el reconeixement de contorns mitjançant cadenes o la cerca de forats convexos, donant així al programador més control sobre el que s'està fent, encara que mai se sap quines altres complicacions haguessin sorgit amb una implementació diferent.

Capítol 9

Conclusió i treballs futurs

Aquest PFC ha tingut un èxit relatiu, doncs es pot assegurar que ha complert els objectius que s'havia marcat, però amb certes dificultats i limitacions. La majoria de les dificultats han vingut donades per la tria de la tècnica de detecció d'objectes mitjançant classificadors en cascada de característiques de tipus Haar. Aquest mètode funciona molt bé amb objectes rics en característiques (que presenten contrastos interns notables) com ara ho són les cares o els logotips. La tasca d'entrenament dels classificadors per reconèixer gestos ha sigut molt feixuga, doncs s'ha tingut un ordinador construint classificadors fins a una setmana seguida, per després comprovar molts cops que el classificador resultant no funcionava. El cas és que els gestos vistos des d'una càmera web de vegades no presenten gaires contrastos característics, encara que sí un perfil concret. Per aconseguir un bon classificador calia doncs buscar els gestos adients que sí presentessin contrastos interns, ajuntant els dits, per exemple, perquè així aparegués una línia fosca entre aquests. El projecte, però, serveix perfectament com a prova de concepte que demostra la viabilitat d'un aplicatiu que permet controlar remotament les presentacions, perquè encara que en un principi s'esperava reconèixer molts gestos, s'han pogut reconèixer els suficients per poder emular les comandes bàsiques d'engegat i parat de la presentació i pas a la següent i anterior diapositiva.
La tècnica de modelat del color de la pell implementada expressament per aquest PFC sí ha donat bons resultats. Era d'esperar que la localització de les àrees on cauen els colors de la pell dins d'un espai de color perceptualment uniforme permetés modelar la seva gamma de colors. La construcció d'un histograma basat en aquestes localitzacions per fer-ho servir més tard com a

taula de cerca dels colors de la pell ha funcionat bé i ha facilitat l'aïllament de les mans i la cara de l'usuari per facilitar el reconeixement dels seus gestos. I no només això, sinó que gràcies a haver aconseguit aïllar la mà de l'usuari de la seva roba s'han pogut fer servir directament les imatges de vídeo on hi apareix fent el gest per extreure'n mostres per entrenar els classificadors. Per tant, podem assegurar que no han hagut sorpreses en la implementació d'aquesta funcionalitat del programa que ha sigut primordial per assolir els objectius que ens marcàvem.

Com a treballs futurs seria interessant provar altres mètodes de reconeixement de gestos en el mateix marc propici aconseguit amb el filtrat de colors i la localització del tronc de l'usuari. També seria interessant fer servir la mateixa idea de disparar events d'entrada al sistema per trobar altres aplicacions al programa, com ara el guiat de robots mòbils o el control remot d'aparells multimèdia. En aquest sentit la instal·lació del programari desenvolupat per aquest PFC a un HTPC (*Home Theater Personal Computer*) amb webcam és perfectament viable, doncs es poden muntar centres multimèdia d'aquest tipus amb Ubuntu i programes com ara el MythTV o Moovida.

Cal remarcar que, al haver fet servir bàsicament l'OpenCV i el Python, que són multiplataforma, l'adaptació futura del codi per altres sistemes operatius diferents del Linux no hauria de comportar majors complicacions que el canvi del mètode de disparat d'events de teclat. Per distribuir l'aplicatiu per Linux o altres sistemes operatius caldria, però, treballar en l'empaquetat del programa perquè els usuaris no hagin d'instal·lar les llibreries requerides manualment.

Finalment voldria comentar que m'agradaria que es publiqués el codi desenvolupat en aquest projecte sota una llicència de programari lliure amb l'assessorament de la Càtedra de Programari Lliure de la UPC, tal i com altres companys han fet amb els seus PFCs. M'ha agradat molt l'experiència de desenvolupar un programa que fa possible una interacció natural amb l'ordinador i potser continuaria millorant-lo des del portal `lafarga.cpl.upc.edu`

Capítol 10

Annexes

10.1 Programari lliure usat en aquest PFC

Aquest PFC s'ha desenvolupat íntegrament amb programari lliure, fent servir principalment:

- Ubuntu 8.04 LTS - `www.ubuntu.com`

Distribució GNU/Linux molt famosa basada en Debian. LTS són les sigles de *Long Term Support*, es tracta, doncs, de la darrera versió amb suport a llarg termini (3 anys) que existia en el moment de començar el PFC.

- OpenCV 2.0 - `opencv.willowgarage.com`

Llibreria de visió per computador que es distribueix sota llicència BSD. S'està convertint en un estàndard *de facto* a molts projectes de visió per computador.

- Python 2.5 - `www.python.org`

Llenguatge de programació d'alt nivell amb el que s'ha implementat la majoria del codi, fent crides a funcions d'OpenCV que disposen, casi en la seva totalitat, d'interfície Python.

- Eclipse Galileo - `www.eclipse.org`

Entorn de desenvolupament integrat amb el que s'ha escrit el codi de l'aplicació. S'ha fet servir el *plugin* PyDev, que facilita el treball amb Python amb el colorejat de la sintaxi, l'autocompletat, autoindentació, marcat d'errors al codi, etc., però sobretot gràcies a un *debugger* que permet, com tots,

executar el codi pas a pas i inspeccionar els valors de les variables així es van executant instruccions. A diferència d'altres, però, té la magnífica possibilitat d'executar codi alhora que s'està debugant, permetent la creació i prova de codi que "veu" les variables que hi ha allà on està aturat el *debugger*.

- CVS - `www.nongnu.org/cvs`

Sistema de control de versions ja antic però que ha cobert perfectament les necessitats d'un sol desenvolupador. Gràcies a aquest programari s'han pogut guardar els canvis en el codi així era rellevant fer-n'hi una còpia i ha permès recuperar-los en els moments adients.

- Glade Interface Designer 3.4.5 - `glade.gnome.org`

Programa de disseny d'interfícies gràfiques d'usuari amb el que s'han fet les finestres. És fàcil de fer servir i té una àmplia comunitat d'usuaris que hi donen suport.

- LyX 1.6.5 - `www.lyx.org`

Processador de textos amb el que s'ha escrit la memòria del PFC i que combina la flexibilitat i poder de LaTeX amb la facilitat d'ús d'una interfície gràfica d'usuari. A diferència dels processadors de text habituals amb aquest programa no cal preocupar-se gaire per l'aspecte del document, doncs és l'aplicatiu qui el construeix.

- Gimp 2.6 - `www.gimp.org`

És un programa de manipulació d'imatges amb el que s'han retocat les figures que es mostren a la memòria.

- GanttProject 2.0.10 - `www.ganttproject.biz`

Aplicatiu d'escriptori que serveix per fer diagrames de Gantt, amb el que s'ha dibuixat la planificació del projecte.

- OpenOffice.org 2.4 - `www.openoffice.org`

Processador de textos que s'ha fet servir com a complement al LyX. Amb ell s'han anat fent esborranys del text de la memòria i s'ha fet el repàs ortogràfic.

Bibliografia

[1] Barczak, A. L. C., and Dadgostar, F. Real-time hand tracking using a set of cooperative classifiers based on haar-like features. In *Research Letters in the Information and Mathematical Sciences* (2005), pp. 29–42.

[2] Baudel, T., and Beaudouin-Lafon, M. Charade: remote control of objects using free-hand gestures. *Commun. ACM 36*, 7 (July 1993), 28–35.

[3] Bhuiyan, M., and Picking, R. Gesture-controlled user interfaces, what have we done and what's next? Tech. rep., Centre for Applied Internet Research (CAIR), Wrexham, UK, 2009.

[4] Bolt, R. A. "put-that-there": Voice and gesture at the graphics interface. In *SIGGRAPH '80: Proceedings of the 7th annual conference on Computer graphics and interactive techniques* (July 1980), vol. 14, ACM Press, pp. 262–270.

[5] Bradski, G., and Kaehler, A. *Learning OpenCV: Computer Vision with the OpenCV Library*, 1st ed. O'Reilly Media, September 2008.

[6] Chen, Q., Georganas, N. D., and Petriu, E. M. Real-time vision-based hand gesture recognition using haar-like features. In *Instrumentation and Measurement Technology Conference Proceedings, 2007. IMTC 2007. IEEE* (2007), pp. 1–6.

[7] Kim, K., Chalidabhongse, T., Harwood, D., and Davis, L. Real-time foreground-background segmentation using codebook model. *Real-Time Imaging 11*, 3 (June 2005), 172–185.

[8] Lienhart, R., Kuranov, A., and Pisarevsky, V. Empirical analysis of detection cascades of boosted classifiers for rapid object detection. *Pattern Recognition* (2003), 297–304.

[9] Lienhart, R., and Maydt, J. An extended set of haar-like features for rapid object detection. In *IEEE ICIP 2002* (2002), vol. 1, pp. 900–903.

[10] Papageorgiou, C. P., Oren, M., and Poggio, T. A general framework for object detection. In *Computer Vision, 1998. Sixth International Conference on* (1998), pp. 555–562.

[11] Rigo, A. Representation-based just-in-time specialization and the psyco prototype for python. In *PEPM '04: Proceedings of the 2004 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation* (New York, NY, USA, 2004), ACM, pp. 15–26.

[12] Sharmam, G. *Digital color imaging handbook*. CRC Press, 2003.

[13] Sonka, M., Hlavac, V., and Boyle, R. *Image Processing, Analysis, and Machine Vision*, 3 ed. CL-Engineering, March 2007.

[14] Tkalcic, M., Tasic, J. F., and Tasic, P. J. F. Colour spaces, 2002. Project for the Digital signal processing course.

[15] Vezhnevets, V., Sazonov, V., and Andreeva, A. A survey on pixel-based skin color detection techniques. In *Proceedings of the GraphiCon 2003* (2003), pp. 85–92.

[16] Viola, P., and Jones, M. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001* (Los Alamitos, CA, USA, April 2001), vol. 1, IEEE Comput. Soc, pp. I–511–I–518.

[17] Wandell, B. A. *Foundations of Vision*. Sinauer Associates, June 1995.

[18] Zarit, B. D., Super, B. J., and Quek, F. K. H. Comparison of five color models in skin pixel classification. In *Recognition, Analysis, and Tracking of Faces and Gestures in Real-Time Systems, 1999. Proceedings. International Workshop on* (1999), pp. 58–63.

Índex alfabètic